



*Príručka programovania v ASM a C
s RTX51 od
Diversant Software
s príkladmi*

*Guide to programming in C and ASM with
RTX51
from
Diversant Software
with examples*



Predslov

Mnoho programátorov sa skôr či neskôr dostane pred problém, ktorý už nemožno elegantným spôsobom riešiť v programovacom jazyku za pomoci cyklického „pooling-u“. Je potrebné vykonávať množstvo úloh súčasne a zároveň je potrebné definovať presné reakčné časy. Na tento problém už dnes existuje riešenie. Je to operačný systém RTX51 Tiny, alebo RTX51 Full. Je vhodný pre embedded systémy a všade tam, kde je potrebné minimalizovať dĺžku vývoja riadiaceho softwaru.

Ing. Eduard Jadroň

Autor

Bez predchádzajúceho písomného povolenia autora nesmie byť ktorákoľvek časť kopírovaná, alebo rozmnožovaná akoukoľvek formou (tlač, fotokópia, mikrofilm, alebo iný postup), zadaná do informačného systému, alebo prenášaná v inej forme alebo inými prostriedkami.

Autor nepreberá záruku za správnosť uvedených materiálov. Predkladané informácie sú zverejnené bez ohľadu na prípadné patenty tretích osôb. Nároky na odškodnenie na základe zmien, chýb alebo vynechania sú vylúčené.

Všetky registrované, alebo iné obchodné známky použité v tejto publikácii sú majetkom ich vlastníkov. Uvedením nie sú spochybnené z uvedeného ich autorské a vlastnícke práva.

Autor

Obsah

1	Tvorba programov v jazyku assembler A51	1
1.1.	Príkazy assembleru A51, direktívy, riadiace príkazy, inštrukcie	1
1.2.	Použitie komentárov a symbolov	1
1.3.	Návestia, operandy, výrazy a operátory	2
1.4.	Deklarácia premenných a konštánt v A51	2
1.5.	Pamäťový podsystem 8051	4
1.6.	Prerušovaci podsystem mikroprocesora 8051	5
1.7.	Deklarácia relokovateľných segmentov v pamäti dát a programu	9
1.8.	Spájanie programov napísaných v A51 a ich preklad	10
1.9.	Zápis inštrukcií assembleru 8051 v jazyku C	12
1.10.	Preklad programovej knižnice rtx51tiny.lib z príkazového riadku	14
1.11.	Preklad programov z príkazového riadku v 64-bitových systémoch	15
1.12.	Ladenie aplikácií napísaných v A51 pomocou µVision	17
1.13.	Programovanie jednoduchých aplikácií v A51 - príklady	21
2	Práca v prostredí µVision	25
2.1.	Postup pri vytváraní projektu	25
2.2.	Tvorba projektu krok za krokom	26
2.3.	Ladenie programu pomocou integrovaného Debugger-a	29
3	Kompilátor C pre architektúru 8051	32
3.1.	Minimálne požiadavky na prekladač	32
3.2.	Moderné požiadavky na prekladač	32
4	Tvorba programov v jazyku C51	33
4.1.	Prehľad používaných architektúr mikroprocesorov	33
4.2.	Prekladače jazyka C pre mikroprocesory	33
4.3.	Príklad zápisu programu v C51	37
4.4.	Deklarácia zložitejších pamäťových štruktúr v C (statická a dynamická alokácia)	38
4.4.1.	Statická alokácia premenných v jazyku C	38
4.4.2.	Dynamická alokácia premenných v jazyku C	38
4.5.	Deklarácia zložitejších pamäťových štruktúr v jazyku C pomocou funkcií RTX51	41
4.6.	Ladenie – simulácia aplikácií v µVision	43
5	Vývojové prostriedky pre tvorbu aplikácií	46
5.1.	Spôsoby tvorby aplikácie	46
5.2.	Prehľad vývojových nástrojov	47
5.3.	Požiadavky na vývojové prostriedky	47
5.4.	Programovací jazyk assembler A51	48
5.5.	Programovací jazyk assembler ASM251 pre procesory rady 80251	48
6	Prehľad vývojových nástrojov	50
6.1.	Grafické vývojové prostredie µVision od firmy KEIL Software	50
6.2.	Grafické vývojové prostredie ProView od Franklin Software	58
6.3.	Grafické vývojové prostredie ADSIM812 od Analog Devices	59
6.4.	Grafické vývojové prostredie Embedded Workbench od IAR Systems	60
6.5.	CodeVision AVR	61
6.6.	Vývojové prostredie Turbo51 pre mikroprocesory 8051	62
6.7.	Vývojové prostredie AS552 od firmy Easy Soft	64
7	Hardware pre vývojové aplikácie	65
7.1.	Hardware 80C51	66
7.2.	Hardware 80C517	67
7.3.	Hardware 80C166	68
7.4.	Hardware MCB2100	69
7.5.	Hardware MCBXC167	69
8	Systémy pracujúce v reálnom čase RTOS	70

8.1.	Opis funkcie RTOS	74
8.2.	Výhody použitia RTOS	75
8.3.	Terminológia RTOS	75
8.4.	Funkcia plánovača – scheduler	76
8.5.	Spôsob práce plánovača v RTX51	79
8.6.	Vzájomná komunikácia medzi úlohami v RTOS	83
9	Systémové požiadavky na operačný systém RTX51	84
9.1.	RTX51 Tiny	84
9.2.	RTX51 Full	84
9.3.	Pamäťový priestor RTX51	85
10	Operačný systém reálneho času RTX51	86
10.1.	Vzájomná komunikácia a synchronizácia medzi úlohami v RTX51	86
10.2.	Mechanizmus prepínania úloh v RTX51	86
10.3.	Riadenie úloh pomocou udalostí v RTX51	87
10.4.	Prerušovací podsystem RTX51	89
11	Komunikácia RTX51 s periférnym rozhraním CAN	91
11.1.	Vlastnosti CAN	91
11.2.	Komunikačný model CAN	92
12	Moderné a perspektívne architektúry mikroprocesorov	98
12.1.	8 bitová architektúra	98
12.2.	16 bitová architektúra	101
12.3.	32 bitová architektúra	102
13	Zoznam funkcií RTX51	104
13.1.	Inicializácia systému RTX51	104
13.2.	Funkcie riadenia úloh	104
13.3.	Funkcie pre obsluhu prerušení	104
13.4.	Funkcie pre synchronizáciu úloh	104
13.5.	Funkcie pre výmenu správ medzi úlohami	105
13.6.	Funkcie pre zdieľanie systémových zdrojov CPU	105
13.7.	Funkcie pre správu externej pamäti	105
13.8.	Funkcie systémového časovača	105
13.9.	Kontrolné a ladiace funkcie	105
14	Programovanie aplikácií s využitím RTX51 Tiny	106
14.1.	Problematika práce s RTX51 Tiny	106
14.2.	Praktické skúsenosti s RTOS RTX51 Tiny	106
15	Príklady programov v ASM, C51 a RTX51 Tiny	108
15.1.	Delay v ASM ako funkcia	108
15.2.	Delay v C51 pomocou cyklu while	108
15.3.	Readkey ako funkcia	108
15.4.	Reset v ASM	109
15.5.	Gregoriánsky kalendár v C51 ako procedúra	109
15.6.	Gregoriánsky kalendár v ASM ako funkcia	110
15.7.	Bitový invertor portu	111
15.8.	Ovládač sériového rozhrania RS232 v C51 bez riadenia prerušením	113
15.9.	Ovládač 232.NET pre multiprocesorovú komunikáciu	114
15.10.	Elektronický sledovač fáz s mikroprocesorom riadený prerušením	119
15.11.	Elektronický sledovač fáz s mikroprocesorom a priamym meraním frekvencie	124
15.12.	Tyristorový regulátor – fázové riadenie usmerňovača	125
15.13.	Fuzzy regulátor v C	126
15.14.	Použitie „overlay“	130
15.15.	Použitie „overlay“ direktívy pri zdieľaní funkcií	130
15.16.	Ovládač RS232 s využitím I/O buffer-a riadeného prerušením	134
15.17.	Univerzálny ovládač komunikačného I ² C rozhrania	135

15.18. Prístup do pamäti E ² PROM 256Byte v DS1624 cez I ² C rozhranie.....	136
15.19. Pamäť E ² PROM s rozhraním I ² C	137
15.19.1. Bloková organizácia pamäti AT24C32/64.....	137
15.19.2. Funkcie jednoduchého zápisu a čítania pre AT24C01 až AT24C16 E ² PROM	138
15.19.3. Funkcie jednoduchého zápisu a čítania pre AT24C32 4kB E ² PROM	139
15.19.4. Funkcie blokového zápisu a čítania pre AT24C32 4kB E ² PROM.....	139
15.20. AD prevodník v 80C552	140
15.20.1. Ovládač pre 10 bitový A/D prevodník v 80C552 v assembleri	140
15.20.2. Ovládač pre 10 bitový A/D prevodník v 80C552 v jazyku C	141
15.21. Použitie a simulácia pamäti E ² PROM v µVision4	141
15.22. Elektronický potenciometer DS1807 s I ² C	142
15.23. Funkcie I ² C pre DS1631 v jazyku C51	144
15.24. Zoznam funkcií pre I ² C rozhranie	152
15.25. Obvod DS1307 - Real Time Clock	153
15.26. Management pamäti v RTX51 Full.....	157
15.27. Management pamäti – odovzdávanie premenných a štruktúr mailbox-om	159
15.28. Ovládanie radiča HD44780 LCD display-a s RTX51 Tiny.	162
15.29. Prevodník portu USB na LPT (USB2LPT).....	169
15.30. PWM kanál s AT89C4051	170
15.31. Univerzálne komunikačné rutiny s CAN rozhraním	174
15.32. Ethernet s DS80C400	176
15.33. ISP rozhranie pre procesory	177
15.34. Práca s ukazovateľom v jazyku C	178
16 Programovanie aplikácií s využitím RTX51 Full.....	179
16.1. Problematika práce s RTX51 Full	179
16.2. Použitie modifikovaných architektur kompatibilných s 8051 s RTX51 Full	179
17 Príklady programov s RTX51 Full	184
17.1. Absolútny a prírastkový diskretný regulátor PSD	184
17.2. 8-bitový PSD regulátor	189
17.3. 32-bitový PSD regulátor	189
17.4. Jednosmerný impulzový menič v znižovacom zapojení STEP-DOWN	190
17.5. Jednosmerný impulzový menič v zvyšovacom zapojení STEP-UP s PFC	193
17.6. Diskretné riadenie 3-fázového usmerňovača s tyristormi	195
17.7. Diskretné riadenie pre tyristorové striedače s rezonančným obvodom	205
17.8. Spracovanie výsledkov meraní na striedači.	208
17.9. Skalárne riadenie asynchrónneho motora pomocou modulácie SWPWM.....	212
17.10. Vektorové riadenie asynchrónneho motora.....	226
17.10.1. Výhody vektorového riadenia SVM (Space Vector Modulation).....	227
17.10.2. Principiálny opis SVM	228
17.10.3. Generovanie výstupného napätia pomocou SVM.....	230
17.10.4. Výpočet vzoriek PWM pri SVM	231
17.10.5. Analýza vytiaženia mikroprocesora C508 pri modulácii SVM.....	235
17.10.6. Zvyšovanie napätia SVM	242
17.11. Zdieľanie sériového kanála viacerými úlohami	246
17.12. Meranie amplitúdy jednosmerného napätia.....	248
17.13. Harmonická analýza analógových veličín s C51 a RTX51.....	251
17.14. Práca s externým pamäťovým priestorom xBANKING	254
18 Inštalácia programu µVision	263
18.1. Inštalácia programu µVision2	263
18.2. Inštalácia programu µVision3, µVision4 a vyšších verzií	263
19 Zoznam bibliografických odkazov	266
20 Zoznam obrázkov	267
21 Zoznam videosekvencií	273
22 Zoznam tabuliek.....	274
23 Zoznam matematických vzťahov	275

1 Tvorba programov v jazyku assembler A51

1.1. Príkazy assembleru A51, direktívy, riadiace príkazy, inštrukcie

Program napísaný v assembleri A51 je tvorený príkazmi, direktívami, riadiacimi príkazmi a inštrukciami. Každý riadok programu môže obsahovať len jeden príkaz, direktívu, inštrukciu. Príkazy napísané v A51 nerozlišuje veľké, alebo malé písmená a môžu začínať na ľubovoľnom riadku a stĺpci. Každý program musí na svojom konci obsahovať direktívu **END**.

Direktívy sú interné príkazy assembleru ktoré nevykonávajú žiadnu činnosť, ale ovplyvňujú len samotný preklad programu. Používajú sa na definíciu konštánt, rezerváciu pamäťového priestoru a pre premenné v programe.

Riadiace príkazy ovplyvňujú preklad programu na základe operátorov, čo umožňuje jednoduché vytvorenie programu ktorý je konfigurovateľný pre rôzne varianty aplikácií.

Inštrukcie sú elementárne príkazy ktoré sú vykonávané CPU. Inštrukcia pozostáva z mnemotechnickej skratky inštrukcie a operandu. Inštrukcia môže mať aj viac operandov. Inštrukcie sú po preklade programu uložené do tzv. „**object**“ súboru, ktorý sa neskôr používa na vytvorenie finálneho programu vo formáte INTEL HEX pre naprogramovanie do pamäti programu CPU.

1.2. Použitie komentárov a symbolov

Komentár je textový reťazec obsahujúci informácie o programe, autorovi a iné informácie. Komentár je nepovinný. Za komentár je považované všetko, čo sa nachádza za znakom „;“.

Symbol je názov pozostávajúci z kombinácie znakov a číslíc ktorý môže reprezentovať číselnú hodnotu, textový blok, alebo meno registra. Symbol nesmie začínať číslom! Vždy musí byť pre definíciu názvu symbolu ako prvé použité písmeno „A“-„Z“, alebo „a“-„z“, prípadne znak „_“, alebo „?“.

1.3. Návestia, operandy, výrazy a operátory

Návestie je symbolická adresa skoku. V programe sa používa ako náhrada za číselné vyjadrenie adresy. Návestie musí mať unikátny názov ukončený znakom „:“ a v programe sa už nesmie nachádzať premenná, alebo návestie s týmto menom. Návestie poskytuje prekladaču a debuggeru dôležité informácie o umiestnení jednotlivých častí programu a dát v pamäti.

Výrazy sú vzájomné kombinácie čísel, znakov, symbolov a operátorov poskytujúcich vypočítané hodnoty ktoré sú používané pri preklade programu. Výsledok výrazu môže byť v prípade A51.EXE maximálne 16 bitová hodnota.

Operandy sú argumenty, alebo výrazy ktoré sú špecifikované assemblerovskými direktívami, alebo inštrukciami. Assemblerovská direktíva vyžaduje operand ktorý je tvorený konštantou, alebo symbolom.

Operátory sú použité na kombinovanie a porovnávanie operandov vo vnútri assemblerovského programu. Nie sú viazané inštrukciami assembleru a teda ich použitím sa nevytvára žiaden programový kód. Operátory poskytujú len základné výpočty a ich hodnoty sú následne odovzdávané prekladanému programu.

1.4. Deklarácia premenných a konštánt v A51

Adresovanie pamäti dát internej a externej pamäti RAM mikroprocesora je vykonávané dvomi spôsobmi:

- Priamym adresovaním, (DATA, BDATA)
- Nepriamym adresovaním, (IDATA, PDATA, XDATA)

Priame adresovanie používajúce pre prístup do pamäti dát pamäťovú triedu **DATA**, a **BIT**. V prvom prípade deklarujeme pamäťové premenné s minimálnou dĺžkou 1 Byte. Druhý prípad umožňuje deklarovať premenné s veľkosťou jedného bitu. Priamym adresovaním je možné v prípade 8051 adresovať pamäťový priestor od adresy 00H do 80H. Bitovo adresovateľný priestor sa nachádza v pamäťovom priestore od adresy 20H do 30H.

mov a,10h ;obsah pamäťovej bunky na adrese 10h skopíruje do registra A

Nepriame adresovanie používajúce pre prístup do pamäti niektorý indexový register ktorý umožňuje prístup do celého adresného priestoru mikroprocesora 8051. Nepriame adresovanie používa register R0, R1 a DPTR. Pamäťová trieda je v tomto prípade **IDATA**, **XDATA** a **CODE** pre pamäť programu. Pre adresovanie externého pamäťového priestoru dát **XDATA** a programu **CODE** je použitá 16 bitová adresa uložená v registri DPTR. Niektoré deriváty napr. AT89C51RD2 využívajú pre prístup väčšieho počtu registrov DPTR, čo v konečnom dôsledku urýchľuje vykonávanie programu.

mov r0,#10h ;register R0 bude ukazovať nepriamo na adresu 10h

mov a,@r0 ;obsah pamäťovej bunky na ktorú ukazuje svojim obsahom R0 zapíše do A

Obrázok 1, Naplnenie súvislého bloku pamäti pomocou nepriameho adresovania

```
29 ;-----
30 Prg1      segment code
31          rseg Prg1
32 ;-----
33 Start:    mov r0,#7fh      ;Pocet byte mazanej pamati v RAM ...
34          clr a             ;Tymto prepisem pamat ...
35 Mazem:    mov @r0,a        ;Teraz vymazem konkretny byte ....
36          djnz r0,Mazem     ;Opakujem skok na Mazem pokiaľ R0<>0
37          nop
38          nop
39          nop
40 ;-----
```

V tomto prípade obsahuje register R0 hodnotu, t.j. počet bajtov ktoré sa použijú v cykle na naplnenie pamäti RAM. Inštrukcia **mov @r0,a** v cykle inicializuje konkrétne oblasti pamäte. Pre prístup do pamäti programu (CODE) sa používa inštrukcia **movc a,@a+dptr** kde sa z adresy pamäťovej bunky vzniknutej súčtom obsahov registrov (A+DPTR) načíta byte a uloží do A. Register DPTR¹ sa používa ako univerzálny ukazovateľ dát v pamäti programu a dát.

Obrázok 2, Príklad správneho zápisu programu v assembleri

```
1  NAME CHAR_IO
2
3  PUBLIC PUTCHAR
4
5  CHAR_ROUTINES SEGMENT CODE      ;definicia programoveho segmentu
6                  RSEG CHAR_ROUTINES ;segment oznacim ako relokovatelny v pamati
7
8  PUTCHAR:        JNB TI,$         ;kontrolujem bit TI t.j. odoslanie znaku
9                  CLR TI          ;znak je uz odoslany, nastavim TI=0
10                 MOV SBUF,A       ;vysielany znak ulozim do vysielacieho registra SBUF
11                 RET
12
13  VAR2            SEGMENT DATA    ;definicia datoveho segmentu
14                  RSEG VAR2       ;segment oznacim ako relokovatelny v pamati
15  DUMMY:          DS 40H          ;vytvorim 64 byte
16
17                  END
18
```

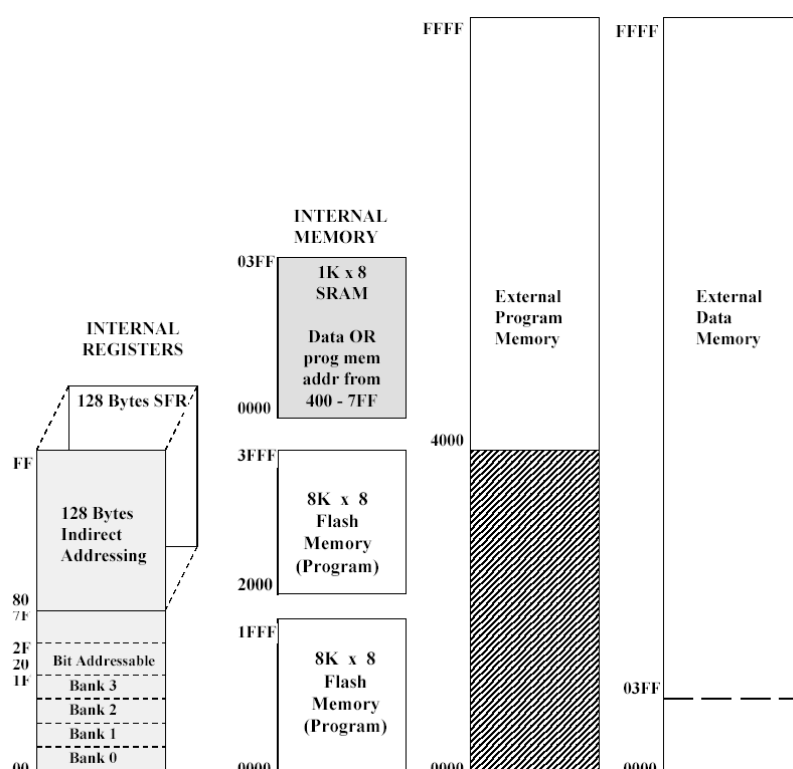
¹ Moderné procesory už majú viacero samostatných registrov DPTR, čo umožňuje rýchlejší prístup k zapísaným údajom v pamätiach procesora.

Vyššie uvedený program obsahuje niekoľko definovaných segmentov. Jeden je programový segment s názvom CHAR_ROUTINES a druhý je dátový segment s názvom VAR2 o veľkosti 40h t.j. 64 Byte. Definované bloky sa pri preklade programu rozmiestnia a optimalizujú v pamäti procesora.

1.5. Pamäťový podsystem 8051

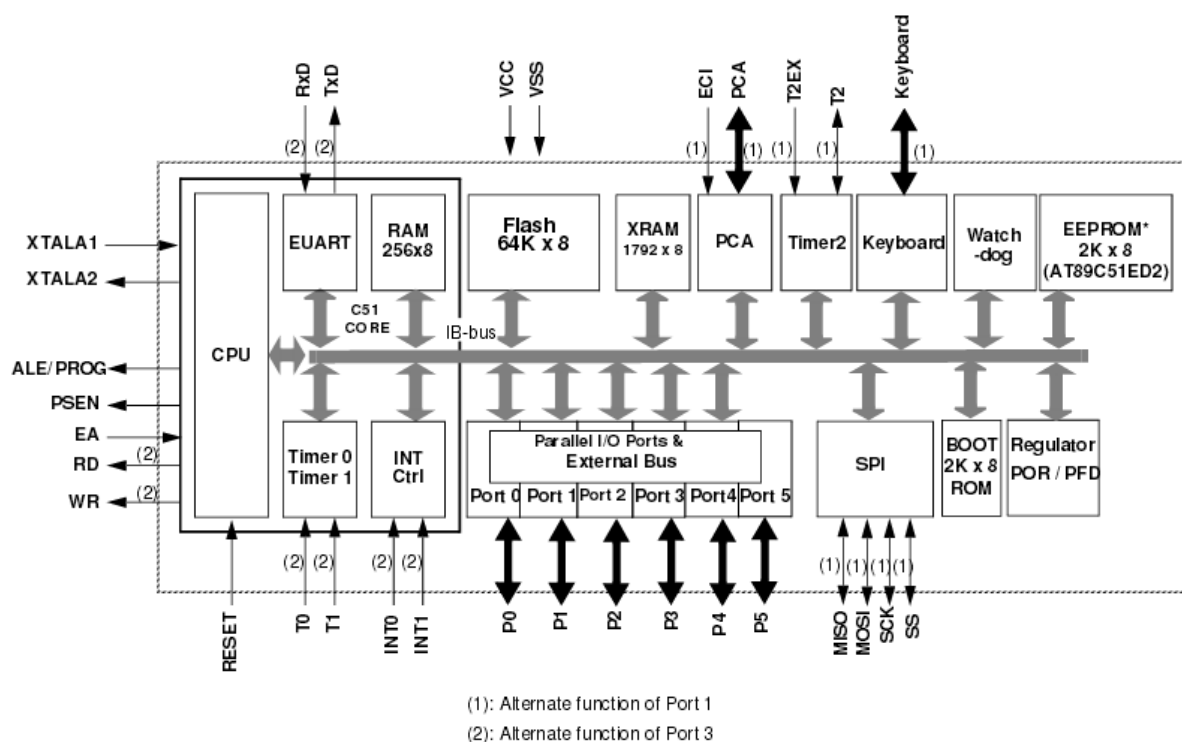
Mikroprocesor rady 8051 obsahuje niekoľko pamäťových priestorov², ktoré je možné využívať programátorom. Obrázok 3 zobrazuje fyzické umiestnenie pamäti v 8051.

Obrázok 3, Pamäťový model a mikroprocesora AT89C51RD2 kompatibilného s 8051



² V súčasnosti sa výrobcovia snažia vyhovieť požiadavkám trhu a do svojich mikroprocesorov integrujú veľké množstvo predtým externých súčiastok do spoločného kremíkového čipu. Napr. A/D, alebo D/A prevodníky, I²C, RS485, PWM, PLA

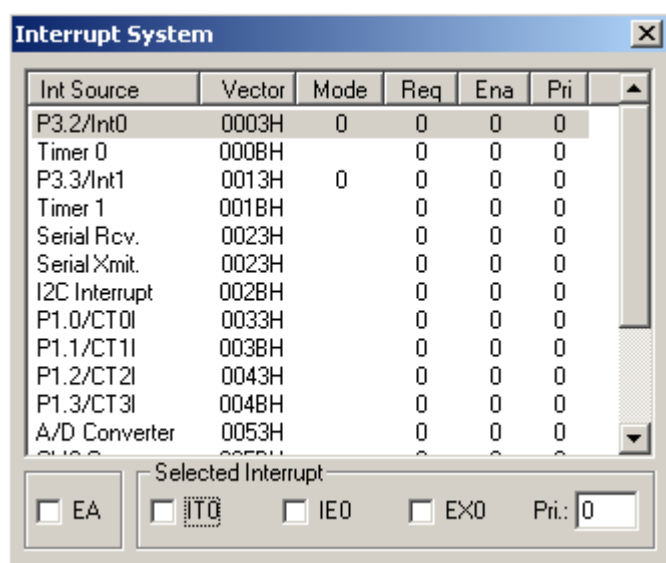
Obrázok 4, Bloková schéma mikroprocesora AT89C51ED2



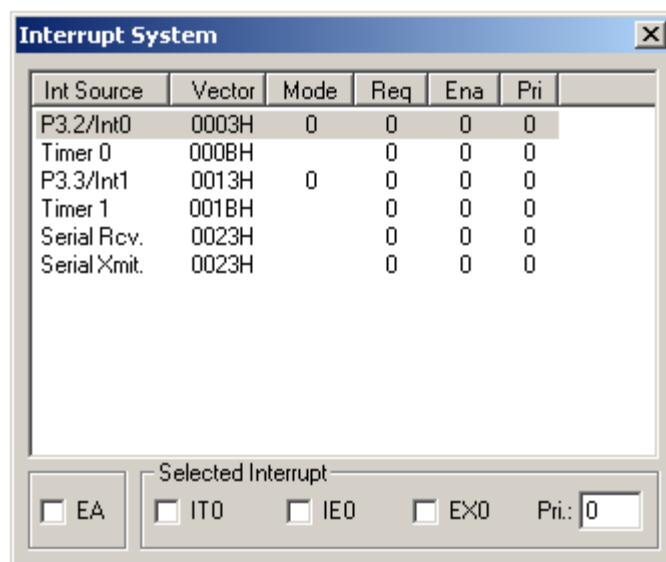
1.6. Prerušovacie podsystem mikroprocesora 8051

Architektúra 8051 umiestňuje svoje vektory prerušenia na absolútne adresy v pamäti programu od adresy 0003H. Vektor prerušenia je v podstate adresa podprogramu, na ktorú mikroprocesor vykoná skok ak je prijaté platné prerušenie. Vykonávaná časť programu sa prijatím prerušenia dočasne pozastaví, na vrchol zásobníka sa uloží návratová adresa nasledujúcej inštrukcie a vykoná sa skok na adresu obsluhy prerušenia t.j. vektor prerušenia. Návrat z prerušenia je vykonaný inštrukciou **RETI**. Táto inštrukcia z vrcholu zásobníka vyzdvihne 2 byte, ktoré použije ako cieľovú adresu skoku, ktorou sa vykoná návrat do programu odkiaľ bol „násilne“ prerušený a pokračuje vo vykonávaní programu.

Obrázok 5, Prerušovacia jednotka mikroprocesora 80C552 v μ Vision4



Obrázok 6, Prerušovacia jednotka mikroprocesora AT89C2051 v μ Vision4



Jednotlivé ladiace okná a ich obsahy sa značne odlišujú od typu použitých procesorov. Veľká variabilita umožňuje programátorovi vyvíjať aplikáciu doslova podľa vopred stanovených požiadaviek.

Obrázok 7, Vzorový príklad programu v macroasembleri³

c:\Omega\data\Dropbox\Záloha\C51\i2c_dsw\ADC_Converter.asm

```

1 ;!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
2 Repeat macro Num
3     mov a,ADCH
4     anl a,#Num
5     rl a
6     rl a
7     ENDM
8 ;!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
9 ;-----
10 public          _ADC_CONVERSION          ;ADC_Conversion(unsigned char)
11 public          CONVERSION               ;Conversion() with ADCON+ADCON
12 ;-----
13 ADCON           data 0C5h
14 ADCH            data 0C6h
15 ;-----
16 ADCS            equ 00001000B           ;Spustim prevod AD prevodnika
17 ADCI            bit acc.4               ;Kontrolujem ci je prevod ukonceny ADCON.4...
18 MASKA           equ 11000000B
19 ;-----
20 Prog_ADC_Converter segment code
21                                rseg Prog_ADC_Converter
22 ;-----
23 _ADC_CONVERSION :
24                                push acc          ;Musi tu byt PUSH ACC, inac to nejde v realnom case
25                                !!!
26                                mov ADCON ,r7      ;Vstupny parameter je cislo kanala pre meranie v R7
27                                orl ADCON ,#ADCS
28 ?ADC_END :          mov a,ADCON
29                                jnb ADCI ,?ADC_END   ;Je prevod ukonceny ???
30                                clr ADCI            ;bit ADCI nastavujem na 0
31                                mov ADCON ,a
32 ;*****
33 CONVERSION :          anl a,#MASKA              ;LSB vysledku je v D7,D6 v ADCON
34                                mov r7,a
35                                Repeat MASKA
36                                mov r6,a          ;MSB vysledku je v R6.1 a R6.0
37                                xch a,r7          ;LSB vysledku je v R7.1 a R7.0
38 ;*****
39                                Repeat ~MASKA     ;Vymaskujem D7 a D6 aby som mal D0 az D5 k
40                                spracovaniu
41 ;*****
42                                orl a,r7           ;LSB R7.1 a R7.0 skombinujem s ADCH0..ADCH5
43                                xch a,r7          ;Vysledok LSB je v R7
44 ;*****
45                                pop acc
46                                ret
47 ;-----
48 ;                                cseg at 0053h      ;os_attach_interrupt(10)
49 ;                                push acc
50 ;                                pop acc
51 ;                                reti
52 ;-----
53                                end

```

³ **Macro** je direktíva prekladača umožňujúca opakujúce sa časti programového kódu opakovane použiť k sprehľadneniu programu napísaného v asembleri.

Tabuľka 1, Zoznam niektorých vektorov prerušení⁴

Číslo prerušenia	Adresa	Symbol	Zdroj prerušenia
0	0003h	$\overline{INT0}$	Externé prerušenie 0
1	000Bh	T0	Prerušenie od časovača T0
2	0013h	$\overline{INT1}$	Externé prerušenie 1
3	001Bh	T1	Prerušenie od časovača T1
4	0023h	ES	Prerušenie od sériovej linky RS232
5	002Bh	T2	Prerušenie od časovača T2
6	0033h	PCA	Prerušenie od PCA
10	0053h	ADC	Prerušenie od 10-bit ADC prevodníka

Tabuľka 2, Prehľad SFR registrov mikroprocesora AT89C51RD2 kompatibilného s 8051

	Bit Addressable	Non Bit addressable							
	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F	
F8h		CH 0000 0000	CCAP0H XXXX XXXX	CCAP1H XXXX XXXX	CCAP2H XXXX XXXX	CCAP3H XXXX XXXX	CCAP4H XXXX XXXX		FFh
F0h	B 0000 0000								F7h
E8h	P5 1111 1111	CL 0000 0000	CCAP0L XXXX XXXX	CCAP1L XXXX XXXX	CCAP2L XXXX XXXX	CCAP3L XXXX XXXX	CCAP4L XXXX XXXX		EFh
E0h	ACC 0000 0000								E7h
D8h	CCON 00X0 0000	CMOD 00XX X000	CCAPM0 X000 0000	CCAPM1 X000 0000	CCAPM2 X000 0000	CCAPM3 X000 0000	CCAPM4 X000 0000		DFh
D0h	PSW 0000 0000	FCON XXXX 0000	ECON XXXX X000	EETIM 0000 0000					D7h
C8h	T2CON 0000 0000	T2MOD XXXX XX00	RCAP2L 0000 0000	RCAP2H 0000 0000	TL2 0000 0000	TH2 0000 0000			CFh
C0h	P4 1111 1111							P5 1111 1111	C7h
B8h	IP X000 000	SADEN 0000 0000							BFh
B0h	P3 1111 1111							IPH X000 0000	B7h
A8h	IE 0000 0000	SADDR 0000 0000							AFh
A0h	P2 1111 1111		AUXR1 XXXX 00X0				WDTRST XXXX XXXX	WDTPRG XXXX X000	A7h
98h	SCON 0000 0000	SBUF XXXX XXXX							9Fh
90h	P1 1111 1111								97h
88h	TCON 0000 0000	TMOD 0000 0000	TL0 0000 0000	TL1 0000 0000	TH0 0000 0000	TH1 0000 0000	AUXR XX0X 1000	CKCON X000 0000	8Fh
80h	P0 1111 1111	SP 0000 0111	DPL 0000 0000	DPH 0000 0000				PCON 00X1 0000	87h
	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F	

 reserved

⁴ Počet vektorov prerušenia závisí od typu procesora. Vektory prerušenia (adresy prerušenia) mikroprocesorov mikroprocesora Dallas Semiconductor DS87C550 nie sú podľa katalógových listov rovnaké, a v prípade použitia náhrady procesora 80C552 priamo DS87C550 je potrebné „presmerovať“ niektoré vektory prerušenia.

1.7. Deklarácia relokovateľných segmentov v pamäti dát a programu

Program napísaný v ľubovoľnom jazyku je zložený z veľkého množstva samostatných procedúr a funkcií ktoré sú rozmiestnené na rôznych miestach v pamäti programu. Pri preklade programu ktorý sa skladá z väčšieho počtu relokovateľných menších častí je výsledný súbor lepšie optimalizovateľný v rámci pamäte mikroprocesora čo v konečnom dôsledku má následný vplyv na rýchlosť vykonávania programu a jeho následnú stabilitu.

Obrázok 8, Príklad zápisu programu v assembleri A51

```
1  LED          BIT  P1.7          ;Deklarácia premennej LED s veľkosťou 1 bit
2
3              CSEG  AT 0000H      ;Preklad programu od adresy 0000H
4              LJMP  START        ;Skok na návštevu START
5
6              CSEG  AT 001BH      ;Preklad programu od adresy 001BH
7              JMP   INT_T1        ;Skok na INT_T1
8
9              CSEG  AT 0100H      ;Preklad programu od adresy 0100H
10  START :      MOV  R0, #07FH      ;Nepriama adresa do R0
11              CLR  A              ;Register A naplním číslom 00h
12  CYKL :      MOV  @R0, A          ;Obsah A zapíšem na adresu uloženú v R0
13              DJNZ R0, CYKL        ;Dekrementujem R0 a pokiaľ R0<>0 skočím na CYKL
14              MOV  SP, #50H        ;Nastavím zásobník na adresu 50H v RAM
15              LCALL INIT
16  LOOP :      JMP  LOOP            ;Vytvorím nekonečný cyklus
17
18  PRG0         SEGMENT CODE        ;Vytváram relokovateľný segment
19              RSEG  PRG0          ;Relokovateľný - premiestniteľný
20  INT_T1 :      INC  COUNTER        ;COUNTER=COUNTER+1
21              CPL  FLAG            ;FLAG=/FLAG
22              CPL  LED
23              RETI
24
25  PRG1         SEGMENT CODE        ;Vytváram relokovateľný segment
26              RSEG  PRG1          ;Relokovateľný - premiestniteľný
27  INIT :      MOV  TMOD, #20H
28              MOV  A, #9CH
29              MOV  TH1, A
30              SETB EA
31              SETB ET1
32              SETB PT1
33              SETB TR1
34              RET
35
36  VAR          SEGMENT DATA        ;Deklarácia segmentu dát v pamäti RAM
37              RSEG  VAR
38  COUNTER :      DS  1              ;Deklarácia pamäťovej premennej s veľkosťou 1 Byte
39
40  VARBIT       SEGMENT DATA        ;Deklarácia segmentu dát v pamäti RAM
41              RSEG  VARBIT
42  FLAG :      DBIT 1              ; Deklarácia pamäťovej premennej s veľkosťou 1 bit
43  END
44
```

Časti programu deklarované ako „SEGMENT CODE“ sú programové segmenty⁵ a „SEGMENT DATA“ sú deklarácie dátového segmentu t.j. deklarácia pamäťovej premennej *COUNTER* údajového typu *byte* a *bitovej* premennej *FLAG*. Použitá direktíva *RSEG* slúži na deklaráciu relokovateľného⁶ segmentu programu, alebo dát.

⁵ Časti programového kódu, alebo pamäti údajov

⁶ Relokovateľnosť (Relocate segment) je možné voľne preložiť ako schopnosť premiestniť program, alebo dát v rámci pamäte bez použitia absolútnej direktívy prekladu od adresy **ORG**, alebo **CSEG AT**.

1.8. Spájanie programov napísaných v A51 a ich preklad

Programy napísané v A51 sa skladajú zo segmentov. Segment je časť programu ktorá môže byť preložená absolútnej adrese, alebo relatívnej. Prvý spôsob sa používa ak je potrebné preložiť program od absolútnej adresy, t.j. obsluha vektorov prerušenia. Druhý je zaujímavejší tým, že neobsahuje adresu na ktorú má byť umiestnený. Hovoríme, že je relokovateľný t.j. premiestniteľný v rozsahu pamäťového programu, alebo dát v priestore mikroprocesora. Linker L51.EXE je zodpovedný za spojenie všetkých segmentov do jedného výsledného, ktorý sa neskôr použije pre naprogramovanie do mikroprocesora. Použitie relokovateľných segmentov prináša nesporné výhody, napriek zdanlivej komplikovanosti. Keďže program pozostáva z niekoľkých menších celkov tzv. segmentov, linker je schopný omnoho lepšie využiť pamäťový priestor pamäti programu, alebo dát a jednotlivé časti segmentu rozmiestni tak, aby zaberali menej pamäti a aby adresy podprogramov a údajových štruktúr sa nachádzali na adrese bez zvyšku deliteľnej dvomi čo zabezpečí rýchlejší prístup mikroprocesora k nim.

Ak potrebujeme spojiť⁷ niekoľko samostatných programov do jedného kompaktného tvaru, tak musíme každý program v assembleri preložiť pomocou programu A51.EXE. Po preklade je vytvorený tzv. „object“ súbor s príponou OBJ. Tento súbor je možné spojiť pomocou špeciálneho programu BL51.EXE nazývaného linker, ktorý „zlinkuje“ všetky súbory do jedného celku. Použitím OH51.EXE je možné preložiť program do spustiteľnej podoby vo formáte INTEL HEX, ktorý sa naprogramuje do mikroprocesora 8051.

Preklad programu:

```
REM This Batch-File generates a Sample-Program for the A51-Assembler
a51.exe asample1.a51 debug xref
a51.exe asample2.a51 debug xref
a51.exe asample3.a51 debug xref
bl51.exe asample1.obj,asample2.obj,asample3.obj to asample precede (var1) ixref
oh51.exe asample
```

⁷ Rovnako spájame aj programy napísané v C, prípadne ich vzájomnú kombináciu napísanú v A51 a C. Vtedy je nevyhnutné použiť direktívu *extern* umožňujúcu sprístupniť jednotlivé procedúry, funkcie a premenné napísané v ľubovoľných jazykoch. Napr. v jazyku C sprístupnenie premennej deklarovanej v A51 bude: *extern unsigned char premenna*; V A51 je nutné vyššie uvedenú premennú uviesť v hlavičke programu ako *public premenna* aby bola viditeľná z A51 v C.

Obrázok 9, Zápis programu v assembleri a jeho pripojenie k jazyku C

```
1  ;-----  
2  public      _Delay          ;Volaie funkcie s parametrom  
3  ;-----  
4  Prog2       segment code    ;Vytvaram cakaciu slucku cca. 300us pre zapis do radica HD pri 0x80  
5              rseg Prog2  
6  ;-----  
7  _Delay:     djnz r7,_Delay   ;Cas do registra R7  
8              ret             ;Vratim sa z rutiny podprogramu  
9  ;-----  
10             end  
11
```

Obrázok 10, Pripojenie programu v assembleri do jazyka C

```
1  #include <reg51.h>  
2  #include <stdio.h>  
3  #include <stdlib.h>  
4  #include <dsw.h>  
5  
6  #include <adresy.inc>  
7  
8  extern Delay (unsigned char);          //Rutinka pre oneskorenie ...  
9  
10 void Init_Serial (void)  
11 {  
12     TMOD |= 0x22 ;           //Nastavim rezim casovaca  
13     SCON |= 0x50 ;           //Nastavim rezim seriového rozhrania  
14     PCON |= 0x80 ;           //Nastavim rezim dvojnásobnej rýchlosti  
15     TH1 = TL1 = 0xFB ;      //Nastavim rezim s danou prenosovou rýchlosťou 19200bps  
16     TR1 = 1 ;               //Spustim casovac ...  
17     Delay (50);             //Chvilku počkam pred inicializáciou ...  
18 }
```

Obrázok 10 názorne ukazuje pripojenie procedúry⁸ Delay() ktorá je napísaná v assembleri a je prilinkovaná ako externá procedúra pomocou *extern Delay(unsigned char)*. V zátvorkách je očakávaný vstupný parameter ktorý následne vstupuje do funkcie ako číslo. Niekedy sa stretávame s tzv. parametrom (*void*), čo v skutočnosti znamená, že do procedúry, alebo funkcie nevstupuje žiadny parameter. Vo vyšších programovacích jazykoch ako je napr. C je vhodné upozorniť prekladač na túto skutočnosť preto, aby sme lepšie využili jeho optimalizačné schopnosti.

⁸ Jazyk C nepozná procedúry, len funkcie. Ak nepotrebujeme návratovú hodnotu funkcie mali by sme použiť **void**, ktorý prekladaču povie, že návratová hodnota funkcie sa nepoužije.

1.9. Zápis inštrukcií assembleru 8051 v jazyku C

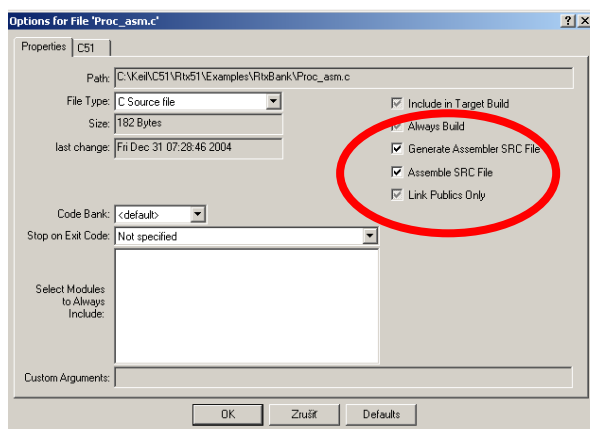
Použitie vyšších programovacích jazykov pri tvorbe aplikácii má svoje nesporné výhody. Aj napriek týmto výhodám potrebujeme niekedy vytvoriť vysoko efektívny, krátky a rýchly program pomocou inštrukcií assembleru mikroprocesora 8051.

Na tieto účely sa používa direktíva prekladača C tzv. **#pragma asm**⁹ a **#pragma endasm**, ktorá umožňuje zápis assembleru do jazyka C. Jediné obmedzenie tohto spôsobu programovania je, že takýto program musí byť zapísaný v samostatnej časti projektu ako samostatný súbor s nastavenými voľbami v okne Options.

Obrázok 11, Príklad zápisu programu v assembleri

```
1 void Delay (unsigned char Dlzka) //Zapis procedury v jazyku C
2 {
3     while ( (-- Dlzka ) != 0x00 );
4 }
5
6 void Proc_asm (void) //Zapis procedury v assembleri 8051
7 {
8     #pragma asm
9     PUSH DPH
10    PUSH DPL
11    INC DPTR
12    MOVC A, @A+DPTR
13    MOV R7, #50h
14    DJNZ R7, $
15    POP DPL
16    POP DPH
17    #pragma endasm
18 }
19
```

Pred samotným prekladom programu je potrebné v okne Options µVision3 zvoliť tieto voľby: Generate Assembler SRC File a Assemble SRC File.



⁹ **#pragma asm** nie je použiteľná v demoverziách.

Výpis súboru PROC_ASM.C po preklade kompilátorom C

```
; \Objects\Proc_asm.SRC generated from: Proc_asm.c
; COMPILER INVOKED BY:
;   C:\Keil\C51\BIN\C51.EXE Proc_asm.c LARGE OMF2 OPTIMIZE(9,SPEED) BROWSE DEBUG
PRINT(\Listings\Proc_asm.lst) SRC(\Objects\Proc_asm.SRC)

NAME    PROC_ASM

?PR?Proc_asm?PROC_ASM          SEGMENT CODE
    PUBLIC  Proc_asm
; void Proc_asm(void)           //Zapis procedury v assembleri 8051

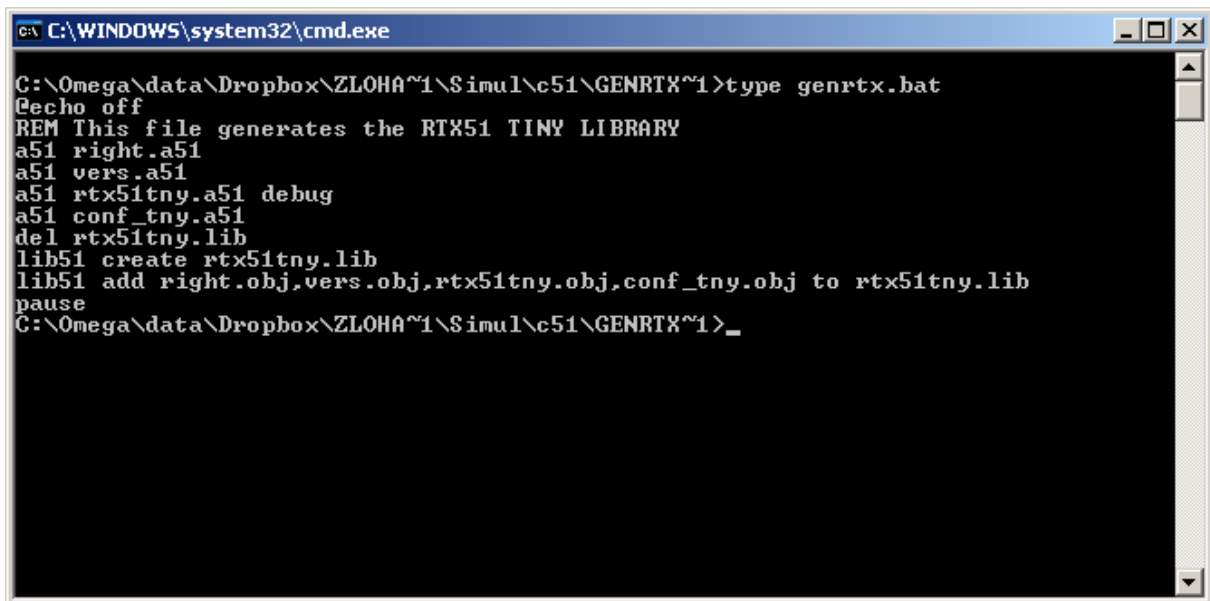
    RSEG ?PR?Proc_asm?PROC_ASM
Proc_asm:
    USING  0
;                               ; SOURCE LINE # 1
; {                               ; SOURCE LINE # 2
; #pragma asm
; PUSH DPH
    PUSH DPH
; PUSH DPL
    PUSH DPL
; INC DPTR
    INC DPTR
; MOVC A,@A+DPTR
    MOVC A,@A+DPTR
; MOV R7,#50h
    MOV R7,#50h
; DJNZ R7,$
    DJNZ R7,$
; POP DPL
    POP DPL
; POP DPH
    POP DPH
; #pragma endasm
; }
;                               ; SOURCE LINE # 13
    RET
; END OF Proc_asm

END
```

1.10. Preklad programovej knižnice rtx51tny.lib z príkazového riadku

Prekladač C51 do verzie 3.40¹⁰ umožňoval len preklad programu z prostredia príkazového riadku (Command line) operačného systému MSDOS verzie 6.22, alebo Windows XP. Výsledkom prekladu je súbor **rtx51tny.lib** ktorý je potrebný pre začlenenie do existujúceho projektu C51 aby bolo možné používať operačný systém RTX51 Tiny¹¹.

Obrázok 12, Preklad programovej knižnice RTX51 Tiny verzie 1.01



```
C:\WINDOWS\system32\cmd.exe
C:\Omega\data\Dropbox\ZLOHA~1\Simul\c51\GENRTX~1>type genrtx.bat
@echo off
REM This file generates the RTX51 TINY LIBRARY
a51 right.a51
a51 vers.a51
a51 rtx51tny.a51 debug
a51 conf_tny.a51
del rtx51tny.lib
lib51 create rtx51tny.lib
lib51 add right.obj,vers.obj,rtx51tny.obj,conf_tny.obj to rtx51tny.lib
pause
C:\Omega\data\Dropbox\ZLOHA~1\Simul\c51\GENRTX~1>_
```

Obrázok 13, Preklad programovej knižnice RTX51 Tiny verzie 2.02

```
@echo off
a51.exe version.a51
a51.exe os_clear_signal.a51
a51.exe os_create.a51
a51.exe os_delete.a51
a51.exe os_reset_interval.a51
a51.exe os_running_task.a51
a51.exe os_send_signal.a51
a51.exe os_set_ready.a51
a51.exe copyright.a51
a51.exe conf_tny.a51
lib51.exe create rtx51tny.lib
lib51.exe add os_clear_signal.obj, os_create.obj, os_delete.obj, os_reset_interval.obj,
os_running_task.obj, os_send_signal.obj, os_set_ready.obj, copyright.obj, conf_tny.obj to rtx51tny.lib
```

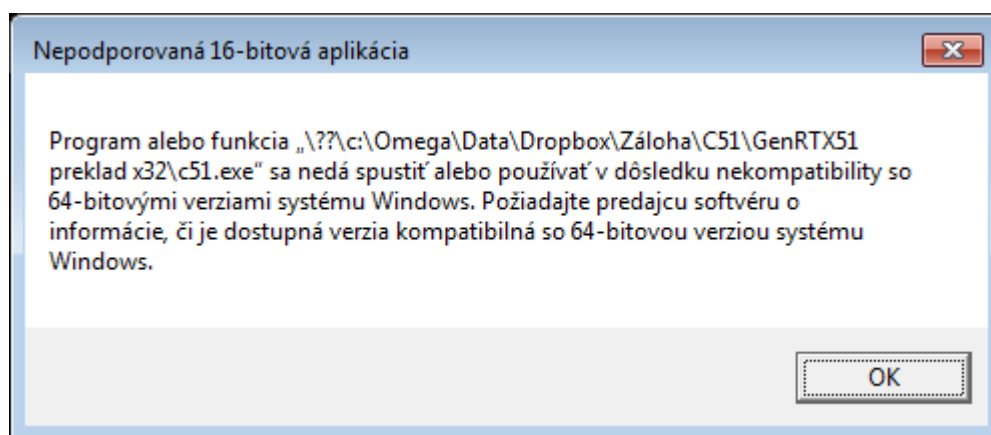
¹⁰ Prekladač C51 verzie 3.40 vyžadoval pre správnu funkciu prítomnosť tzv. dongle (hardware-ového kľúča) ako ochranu proti nelegálnemu kopírovaniu. Ak bol tento „dongle“ na porte LPT1, LPT2 pri zapojenej tlačiarne, dochádzalo pravidelne vždy pri kompilácii programu napísaného ho v C k „RESET-u“ tlačiarne, alebo nesprávnej funkcii zariadení ktoré boli pripojené na paralelnom LPT porte. Program C51.EXE cez LPT1, alebo LPT2 port zisťoval prítomnosť hardware-ového kľúča.

¹¹ RTX51 Tiny 2.02 je oproti verzii 1.01 podstatne vylepšený čím sa dosiahlo vysokej stability operačného systému.

1.11. Preklad programov z príkazového riadku v 64-bitových systémoch

V niektorých prípadoch je potrebné preložiť program napísaný v jazyku C bez vývojového prostredia μ Vision priamo v konzole operačného systému Windows.¹² Problémy spôsobuje hlavne prostredie 64-bitových¹³ operačných systémov, ktoré neumožňuje spustenie prekladača jazyka C **c51.exe**, **l51.exe**, alebo **oh51.exe** verzie 3.40 určenej ešte pre 16-bitový operačný systém MS-DOS. Preto je potrebné z adresára **c:\keil\c51\bin** legálnej distribúcie KEIL μ Vision¹⁴ skopírovať do zvoleného adresára nasledujúce súbory: a51.exe, ax51.exe, cx51.exe, lx51.exe, ohx51.exe, rtx51tny.lib, c51fps.lib, c51fpc.lib, c51fpl.lib, reg51.h, stdio.h, stdlib.h, rtx51tny.h, l51.dll, tools.ini.¹⁵

Obrázok 14, Výpis chybového hlásenia o nepodporovanej 16-bitovej aplikácii



Prvá možnosť ako je možné preložiť program v 64-bitovom systéme je nainštalovanie virtuálneho počítača ako je napr. VirtualBox, alebo VMware v prostredí 64-bitového systému, kde je možné inštalovať 16-bitové prostredie MS-DOS a preložiť program. Druhá možnosť už bola načrtnutá a spočíva v skopírovaní niektorých súborov ktoré sú obsiahnuté v legálnej inštalácii μ Vision. Dávkovým súborom potom môžeme preložiť program a pomocou ohx51.exe exportovať do hex súboru ktorá je pomocou špeciálneho programátora, alebo pomocou ISP naprogramovaná do mikroprocesora.

¹² Windows XP 64-bit, Windows 7 64-bit, Windows 8.1 64-bit, Windows 10 64-bit

¹³ 32-bitové operačné systémy môžu adresovať maximálne 4GB fyzickej pamäti RAM, u 64-bitových systémoch už reálne obmedzenie v podobe približne **16,8 miliónov terabyte** nie je.

¹⁴ Vyskúšané a funkčné vo verzii Keil μ Vision 9.54a

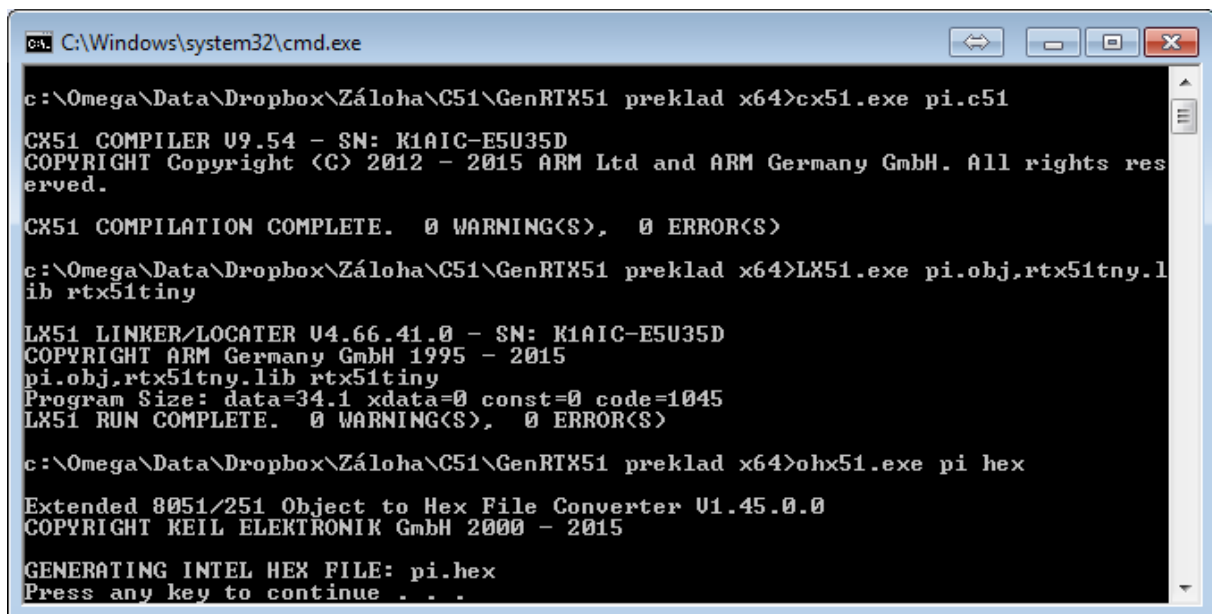
¹⁵ Súbor tools.ini obsahuje registračné údaje o Keil μ Vision ako je napr. S/N, alebo LIC.

Obrázok 15, Preklad programu pomocou dávkového súboru

```
@echo on
cx51.exe pi.c51
lx51.exe pi.obj,rtx51tiny.lib rtx51tiny
ohx51.exe pi hex
@echo off
del *.lst *.m51 *.
pause
```

Dávkový súbor môžeme pomenovať napr. **pi.bat**¹⁶ a v tomto tvare je spustiteľný aj v prostredí 64-bitového operačného systému.

Obrázok 16, Výpis prekladu programu v 64-bitovom systéme



```
C:\Windows\system32\cmd.exe

c:\Omega\Data\Dropbox\Záloha\C51\GenRTX51 preklad x64>cx51.exe pi.c51

CX51 COMPILER U9.54 - SN: K1AIC-E5U35D
COPYRIGHT Copyright (C) 2012 - 2015 ARM Ltd and ARM Germany GmbH. All rights reserved.

CX51 COMPILATION COMPLETE.  0 WARNING(S),  0 ERROR(S)

c:\Omega\Data\Dropbox\Záloha\C51\GenRTX51 preklad x64>LX51.exe pi.obj,rtx51tiny.lib rtx51tiny

LX51 LINKER/LOCATER U4.66.41.0 - SN: K1AIC-E5U35D
COPYRIGHT ARM Germany GmbH 1995 - 2015
pi.obj,rtx51tiny.lib rtx51tiny
Program Size: data=34.1 xdata=0 const=0 code=1045
LX51 RUN COMPLETE.  0 WARNING(S),  0 ERROR(S)

c:\Omega\Data\Dropbox\Záloha\C51\GenRTX51 preklad x64>ohx51.exe pi hex

Extended 8051/251 Object to Hex File Converter U1.45.0.0
COPYRIGHT KEIL ELEKTRONIK GmbH 2000 - 2015

GENERATING INTEL HEX FILE: pi.hex
Press any key to continue . . .
```

Z vyššie uvedeného obrázku je vidieť, že preklad programu bol úspešný a bol vygenerovaný hex súbor, ktorý už môžeme naprogramovať do mikroprocesora.

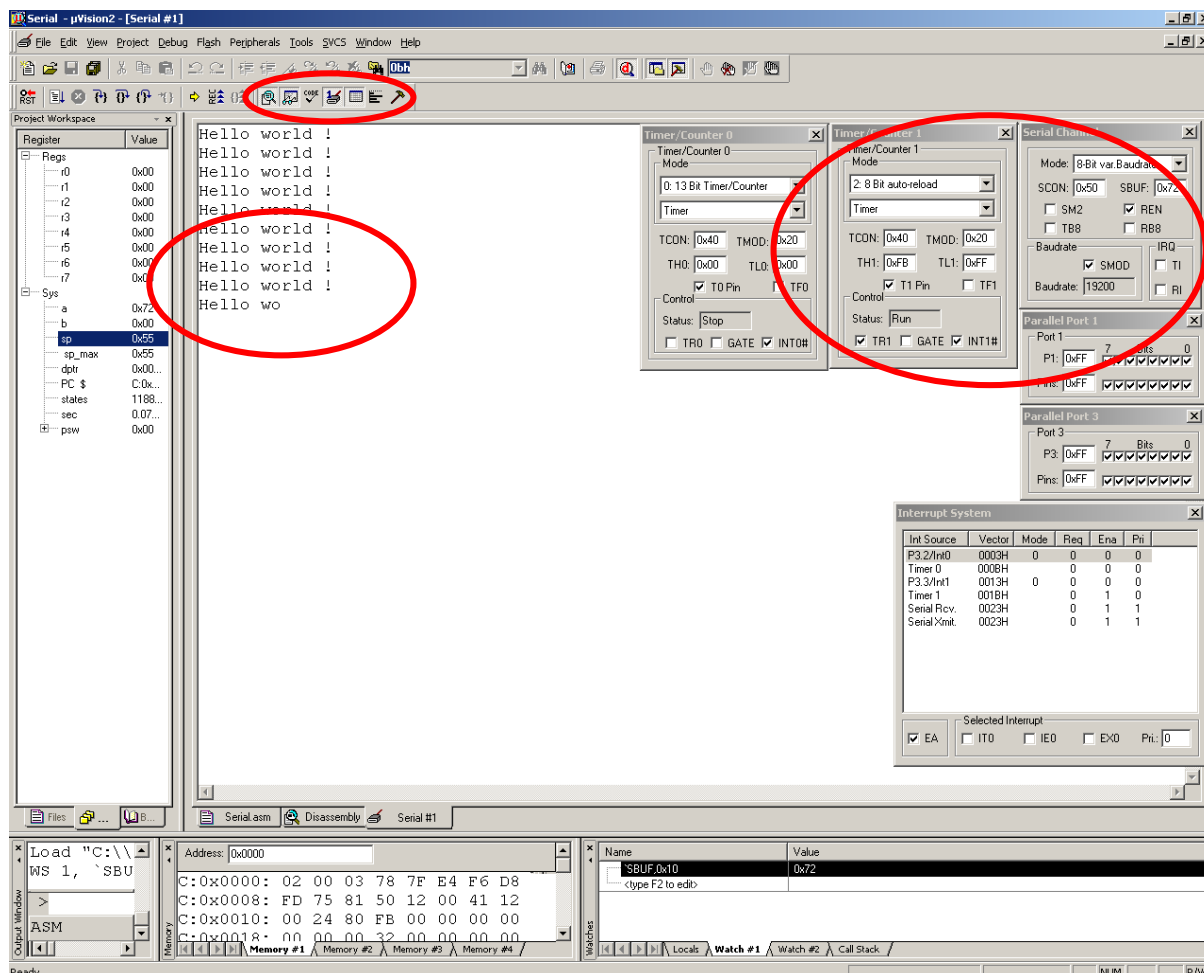
¹⁶ Niektoré antivírusové programy upozorňujú pri spustení na možnú nebezpečnosť pri použití dávkového súboru. Odporúčame preto radšej tieto hlásenia antivírusového vypnúť, aby neustále nevypisoval možnosť ohrozenia PC, čo je nesprávny oznam pre užívateľa.

1.12. Ladenie aplikácií napísaných v A51 pomocou µVision

Každý program sa musí po napísaní v počiatočnej fáze odladiť – odskúšať. Na túto činnosť existuje množstvo nástrojov líšiacich sa možnosťami a samozrejme aj cenou. Medzi najjednoduchšie môžeme zaradiť tzv. softwarové (programové) simulátory, ktoré síce umožnia daný program odladiť, ale s väčšími, alebo menšími nepresnosťami. Lepšie výsledky nám dávajú tzv. emulátory, ktoré sú v osadené cieľovým mikroprocesorom v ktorom je vykonávaný odladovaný program.

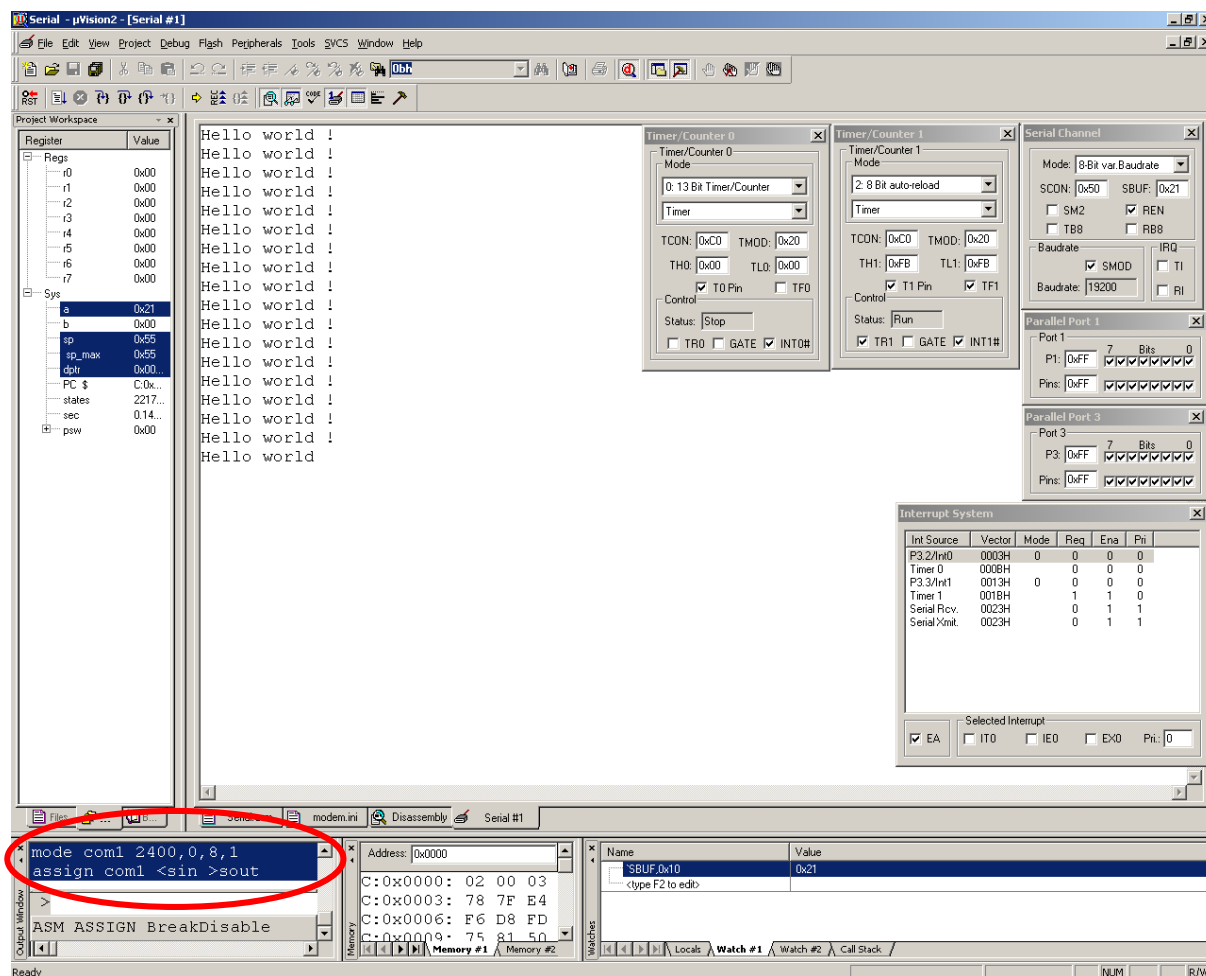
Pre naše potreby bude postačovať ladiaci program – debugger ktorý je súčasťou prostredia µVision. Pre úspešné odladenie programu je potrebné v µVision3 zvoliť v menu programu položku „Serial Window #1” a ak je program správne napísaný tak po jeho spustení začne vypisovať celú komunikáciu mikroprocesora na virtuálny¹⁷ sériový kanál prostredia µVision.

Obrázok 17, Odladovaný program sériového rozhrania v µVision



¹⁷ Výpis musí byť presmerovaný len na port COM1 až COM4 t.j. fyzický port v PC, ináč nepracuje správne! Porty COM1 až COM4 (3F8h, 3E8h, 2F8h, 2E8h) je možné použiť ako obojsmerné porty pri ladení aplikácií s rôznym hardware pomocou funkcií `scanf()` a `printf()`, prípadne so sériovým registrom mikroprocesora SBUF.

Obrázok 18, Nastavenie μ Vision pre presmerovanie sériového kanála na port COM.



Ak sa pri odlaďovaní¹⁸ použije možnosť presmerovanie sériového kanála prostredia μ Vision na porty PC, celá komunikácia mikroprocesora 8051 bude zachytávaná, zobrazovaná na obrazovke a zároveň odosielaná na porty COM1 až COM4. Je nutné upozorniť, že každá operácia čítania, alebo zápisu s registrom SBUF spôsobí zmenu na príslušných portoch. Určitou nevýhodou tohto programu je obmedzený rozsah komunikačných portov obmedzujúcich¹⁹ sa len na fyzické porty COM1 až COM4. Aj napriek tejto nevýhode je možno pripojiť napríklad modem, GSM mobil, sériovú tlačiareň, čítačku čiarových kódov a iných zariadení ktoré komunikujú cez sériové rozhranie RS232.

Pre nastavenie parametrov sériového kanála v prostredí μ Vision sa používa príkaz MODE ktorý poznáme z prostredia MSDOS-u s parametrami.

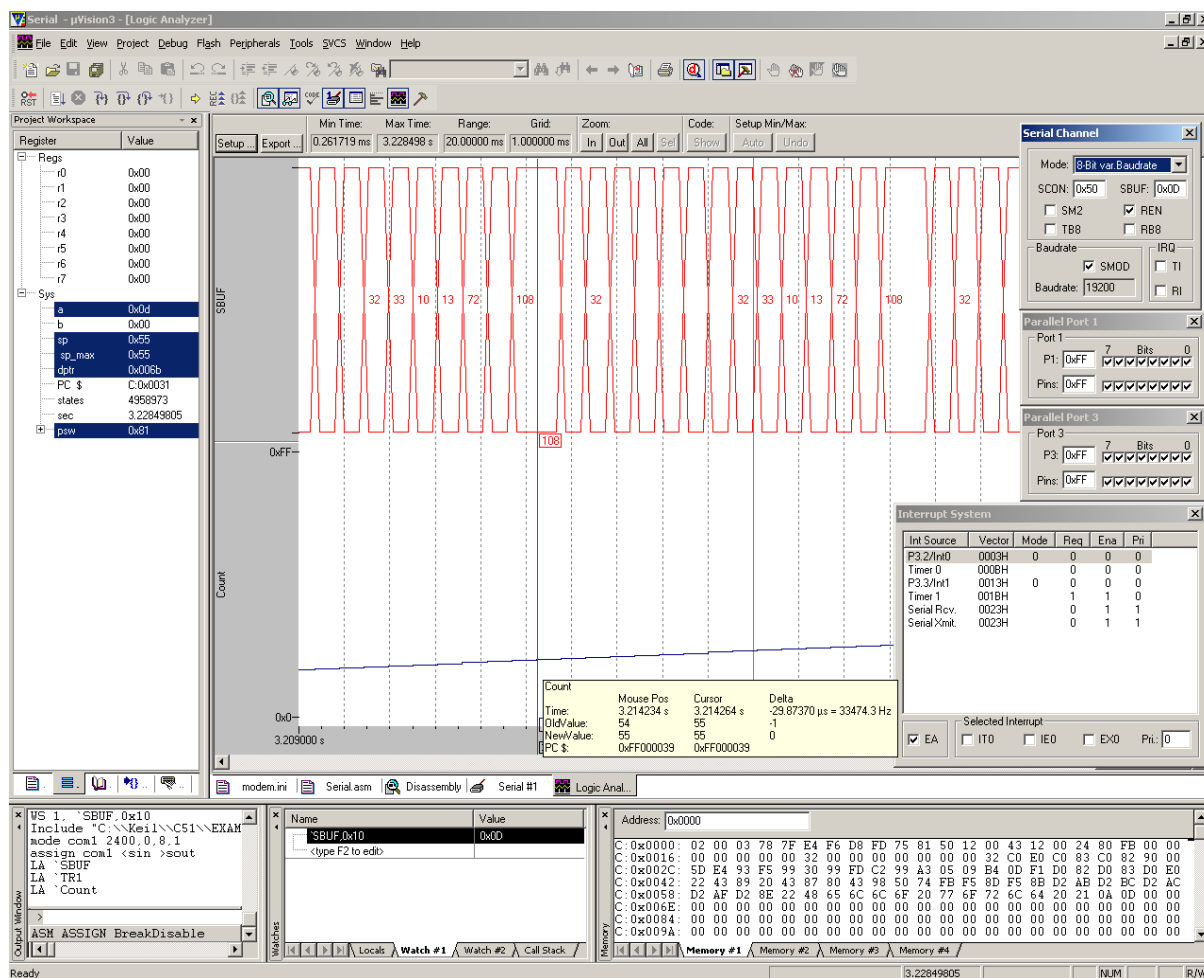
¹⁸ Pre správnu činnosť funkcie **printf()** je nutné po inicializácii sériového kanála nastaviť bit **TI=1**. Príznakový bit **TI=0** je potrebné nastaviť len vtedy ak používame pri prenose údajov prerušenie od sériového kanála pre sériový prenos s adresou obsluhy prerušenia na adrese 0023H.

¹⁹ Nové PC majú maximálne len jeden fyzický port COM1, ktorý tam je len kvôli spätnej kompatibilitate s existujúcim hardware a software.

MODE COM1 2400,0,8,1 (2400 – prenosová rýchlosť, bez parity, 8 bitov, 1 stop bit)

ASSIGN COM1 <sin>sout (Priradenie vstupu/výstupu portu COM1)

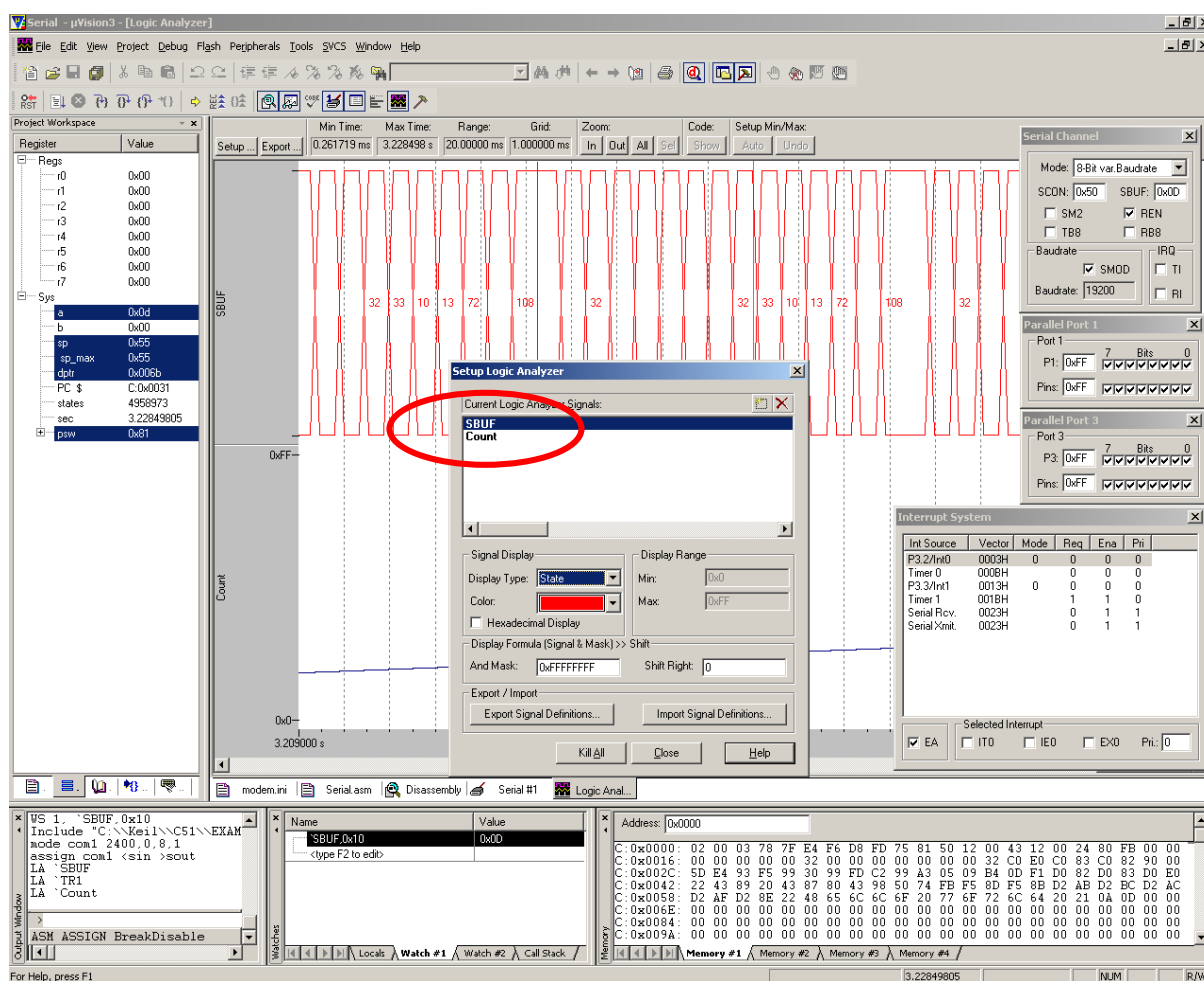
Obrázok 19, µVision3 a využitie funkcie Logic Analyzer - osciloskop



Zobrazenie časového priebehu logických úrovní na jednotlivých portoch mikroprocesora 8051 je bez použitia drahých logických analyzátorov veľmi komplikované a náročné. Túto náročnú operáciu je možné práve v softwarových simulátoroch veľmi dobre integrovať. Pri odladovaní programu môže programátor sledovať priebehy signálov na portoch mikroprocesora. Túto funkciu je možné použiť len od µVision3²⁰ od firmy Keil.

²⁰ Užívateľ môže od tejto verzie použiť v sekcii „Option for Target“ tzv. „Initialization File“ ktorý je v podstate program C, ktorý môže generovať rôzne analógové, alebo číslkové priebehy t.j. signály pre hlavný program. Veľkou výhodou je, že v tomto programe je možné pracovať so všetkými premennými ktoré sú deklarované v hlavnom programe ktorý odladujeme. Problém môžu spôsobovať procesory s viacerými sériovými rozhraniami RS232, CAN, I²C, kde je potom potrebné zmeniť názov simulovaného kanála z SIN na SIN0.

Obrázok 20, Nastavenie funkcie Logic Analyzer - osciloskop



Nastavenie funkcie osciloskopu je možné vykonať v okne „Setup Logic Analyzer“. V tomto okne je možné nastaviť typ premennej²¹, portu a spôsob ich reprezentovaného zobrazenia na obrazovke. Zároveň je možné všetky nastavenia importovať, exportovať, alebo maskovať a zapísať do súboru pre neskoršie použitie. Zobrazované názvy signálov a parametre sériového rozhrania je možné vidieť aj v ľavom dolnom rohu prostredia.

²¹ Aby bola viditeľná na osciloskope musí byť premenná deklarovaná ako globálna. Lokálne deklarované premenné a premenné ktoré sú zapuzdrené vo funkciách nie je možné zobrazovať.

1.13. Programovanie jednoduchých aplikácií v A51 - príklady

Príklad: Napíšte program v A51, ktorý bude vykonávať reguláciu pohonu jednosmerného motora s cudzím budením pomocou PSD regulátora (*prírastkového*)²², kde skutočná hodnota nameranej veličiny sa bude nachádzať na porte P1, žiadaná hodnota na P3 a akčná veličina na porte P2. Všetky premenné, konštanty a vstupné hodnoty budú kvôli jednoduchosti programu 8 bitové celočíselné so znamienkom (*signed char*)²³:

$$y_N = y_{N-1} + A \cdot e_N - B \cdot e_{N-1} + C \cdot e_{N-2} \quad (1.)$$

Kde **A**, **B**, **C** sú konštanty a e_N je odchýlka žiadanej a skutočnej regulovanej veličiny.

$$A = K \cdot \left(1 + \frac{\Delta T}{2T_i} + \frac{T_d}{\Delta T} \right) \quad (2.)$$

$$B = K \cdot \left(1 - \frac{\Delta T}{2T_i} + \frac{2T_d}{\Delta T} \right) \quad (3.)$$

$$C = K \cdot \left(\frac{T_d}{\Delta T} \right) \quad (4.)$$

V kapitole 17.2 a 17.3 je uvedený diskretný PSD regulátor naprogramovaný v prostredí operačného systému RTX51 s 32 bitovou aritmetikou, kde jednotlivé premenné majú reálny údajový typ. V praxi sa snažíme reálnu aritmetiku nahradiť celočíselnou, ktorá je v mikroprocesoroch vypočítaná omnoho rýchlejšie.

²² Prírastkový regulátor sa od absolútneho odlišuje výrazne v rýchlosti výpočtu, pretože počíta v algoritme len konkrétne prírastky z regulačnej odchýlky a nie súčty jednotlivých súm, ktoré sa vo výpočte pripočítavajú k zásahu na výstupe regulátora. Pre urýchlenie výpočtu je vhodné pri reálnej aritmetike použiť tzv. tabuľkový spôsob, ktorý spočíva v uložení výsledkov výpočtov do tabuľky, ktoré neskôr používame v algoritme.

²³ Použitý údajový typ bol len pre verziu napísanú v jazyku assembler, pre jazyk C je použitý celočíselný údajový typ **int**, alebo reálny údajový typ **float**.

Príklad zápisu programu:

c:\Omega\data\Dropbox\Záloha\C51\PSD8bit\PSD_8bit_asm.asm

```
1 ;-----
2 ?PR?main?PSD8 SEGMENT CODE
3 ?DT?PSD8 SEGMENT DATA
4 ;-----
5 RSEG ?DT?PSD8
6 ;-----
7 KA: DS 1
8 KB: DS 1
9 KC: DS 1
10 En: DS 1
11 Is: DS 1
12 Iz: DS 1
13 Yn: DS 1
14 En1: DS 1
15 En2: DS 1
16 Yn1: DS 1
17 ;-----
18 RSEG ?PR?main?PSD8
19 USING 0
20 ;-----
21 MOV KA, #01H
22 MOV KB, #02H
23 MOV KC, #03H
24 Cykl:
25 MOV Iz, P1
26 MOV Is, P3
27 ;-----
28 ; En=Iz-Is;
29 ;-----
30 CLR C
31 MOV A, Iz
32 SUBB A, Is
33 MOV En, A
34 ;-----
35 ; Yn=Yn1+(KA*En)-(KB*En1)+(KC*En2);
36 ;-----
37 MOV A, KA
38 MOV B, En
39 MUL AB
40 ADD A, Yn1
41 MOV R7, A
42 MOV A, KB
43 MOV B, En1
44 MUL AB
45 MOV R6, A
46 CLR C
47 MOV A, R7
48 SUBB A, R6
49 MOV R7, A
50 MOV A, KC
51 MOV B, En2
52 MUL AB
53 ADD A, R7
54 MOV Yn, A
55 ;-----
56 MOV Yn1, A
57 MOV En2, En1
58 MOV En1, En
59 SJMP Cykl
60 ;-----
61 END
62
```

Príklad: Napíšte program typu „reklama“ ktorý bude v jazyku assembler periodicky vypisovať na porte P1 údaje nachádzajúce sa v tabuľke. Tabuľka obsahuje tzv. spínací plán LED diód pripojených na porte P1. Koniec tabuľky bude označený znakom '*'.

```
1 ;-----
2 BANK      macro    N
3             mov     psw, #(&N shl 3)
4             endm
5 ;-----
6             cseg at 0000h
7             jmp     Start
8 ;-----
9 Prg0       segment code
10            rseg    Prg0
11 ;-----
12 Tabulka :  db      10000000b,01000000b,00100000b,00010000b
13            db      00001000b,00000100b,00000010b,00000001b
14            db      00000001b,00000010b,00000100b,00001000b
15            db      00010000b,00100000b,01000000b,10000000b
16            db      11001100b,00110011b,11001100b,00110011b
17            db      11001100b,00110011b,11001100b,00110011b
18            db      11110000b,00001111b,11110000b,00001111b
19            db      11110000b,00001111b,11110000b,00001111b
20            db      11111111b,00000000b,00000000b,00000000b
21            db      11111111b,00000000b,00000000b,00000000b
22            db      11111111b,00000000b,00000000b,00000000b
23            db      11111111b,00000000b,00000000b,00000000b
24            db      11111111b,11111111b,11111111b,11111111b
25            db      11111111b,11111111b,11111111b,11111111b
26            db      11111111b,11111111b,11111111b,11111111b
27            db      11111111b,11111111b,11111111b,11111111b
28            db      '*'
29 ;-----
30 Prg1       segment code
31            rseg    Prg1
32 ;-----
33 Start :    mov     r0,#7fh          ;Pocet byte mazanej pamati v RAM ...
34            clr     a                ;Tymto prepisem pamat ...
35 Mazem :    mov     @r0,a            ;Teraz vymazem konkretny byte ....
36            djnz    r0,Mazem         ;Opakujem skok na Mazem pokiaľ R0<>0
37 Zobrazuj : mov     dptr,#Tabulka    ;Kde je Tabulka ???
38 Cyklus :   clr     a
39            movc    a,@a+dptr        ;Nacitam prvý byte z Tabulky
40            cjne    a,'#*',Pokracuj  ;Ak A<>'*' Pokracuj
41            jmp     Zobrazuj
42 ;-----
43 Prg2       segment code
44            rseg    Prg2
45 ;-----
46 Pokracuj : mov     P1,a             ;Zobrazim na LED pripravku
47            call    Delay            ;Oneskorim to ....
48            inc     dptr             ;Posuniem sa na ďalší byte v Tabulke
49            cjne    a,'#*',Cyklus    ;Ak A<>'*' tak zobrazujem !!!
50            jmp     Zobrazuj
51 ;-----
52 Prg3       segment code
53            rseg    Prg3
54 ;-----
55 Delay :    BANK    3                ;Prepnem sa na inú banku registrov
56            mov     r0,#0FFh         ;Oneskorim to o asi 255*255 us
57            mov     r1,#0FFh
58 CCC :      djnz    r1,$
59            djnz    r0,CCC
60            BANK    0                ;Vratim sa na pôvodnú banku registrov
61            ret
62 ;-----
63
64            end
```

Príklad: Napíšte program, ktorý bude cez sériové rozhranie RS232 vysielat' rýchlosťou 19200b.s⁻¹ textový reťazec "Hello world !". Časti opakujúceho sa kódu napíšte pomocou makra. V programe definujte premenné umiestnené v programových, dátových segmentoch.

C:\Omega\data\Dropbox\Záloha\C51\xxx\serial.asm

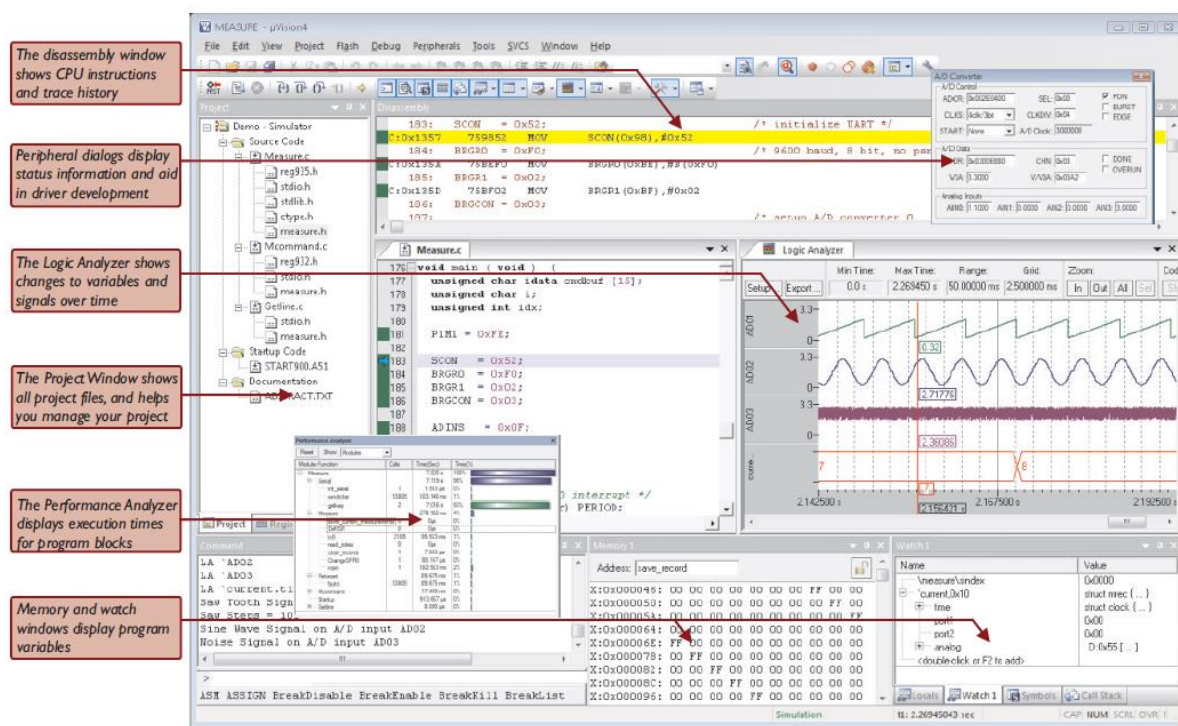
```
1  Vars          segment data
2  Txt           segment code
3  Prog          segment code
4  Initial       segment code
5  Send          segment code
6  Koniec        equ 0dh
7  ;-----
8  Save          macro
9                irp reg <acc,dph,dpl>
10               push reg
11               endm
12               endm
13 ;-----
14 Restore       macro
15               irp reg <dpl,dph,acc>
16               pop reg
17               endm
18               endm
19 ;-----
20              rseg Vars
21 Znak :         ds 1
22 ;-----
23              rseg Txt
24 Text :         db "Hello world !",0ah,Koniec
25 ;-----
26              cseg at 0000h
27              jmp Start
28 ;-----
29              cseg at 001bh
30              reti
31 ;-----
32              cseg at 0023h
33 End232 :       reti
34 ;-----
35              rseg Initial
36 ;-----
37 Init :         orl tmod,#20h
38               orl pcon,#80h
39               orl scon,#50h
40               mov a,#0fbh
41               mov thl,a
42               mov tll,a
43               setb etl
44               setb ps
45               setb es
46               setb ea
47               setb trl
48               ret
49 ;-----
50              rseg Send
51 ;-----
52 Posli :        Save
53               mov dptr,#Text
54 Cykl :         clr a
55               movc a,@a+dptr
56               mov sbuf,a
57               jnb ti,$
58               clr ti
59               inc dptr
60               cjne a,#Koniec,Cykl
61               Restore
62               ret
63 ;-----
64               end
```

2 Práca v prostredí µVision

Cieľom tejto kapitoly bude čitateľa oboznámiť s filozofiou tvorby projektu v prostredí µVision3. Spôsob tvorby projektu sa diametrálne líši od spôsobu pri práci v AS552, ale minimálne v iných²⁴ IDE prostrediach.

Projekt je súhrn všetkých súborov, knižníc ktoré sú potrebné pre vytvorenie výsledného spustiteľného tvaru HEX, BIN v mikroprocesore 8051, alebo knižnice s príponou LIB.

Obrázok 21, Vzorový príklad projektu µVision4



The µVision development platform is easy to use and it helps you to quickly create embedded programs that work. The µVision editor and debugger are integrated in a single application that provides a seamless embedded project development environment for editing, simulating, Flash programming and testing in target hardware.

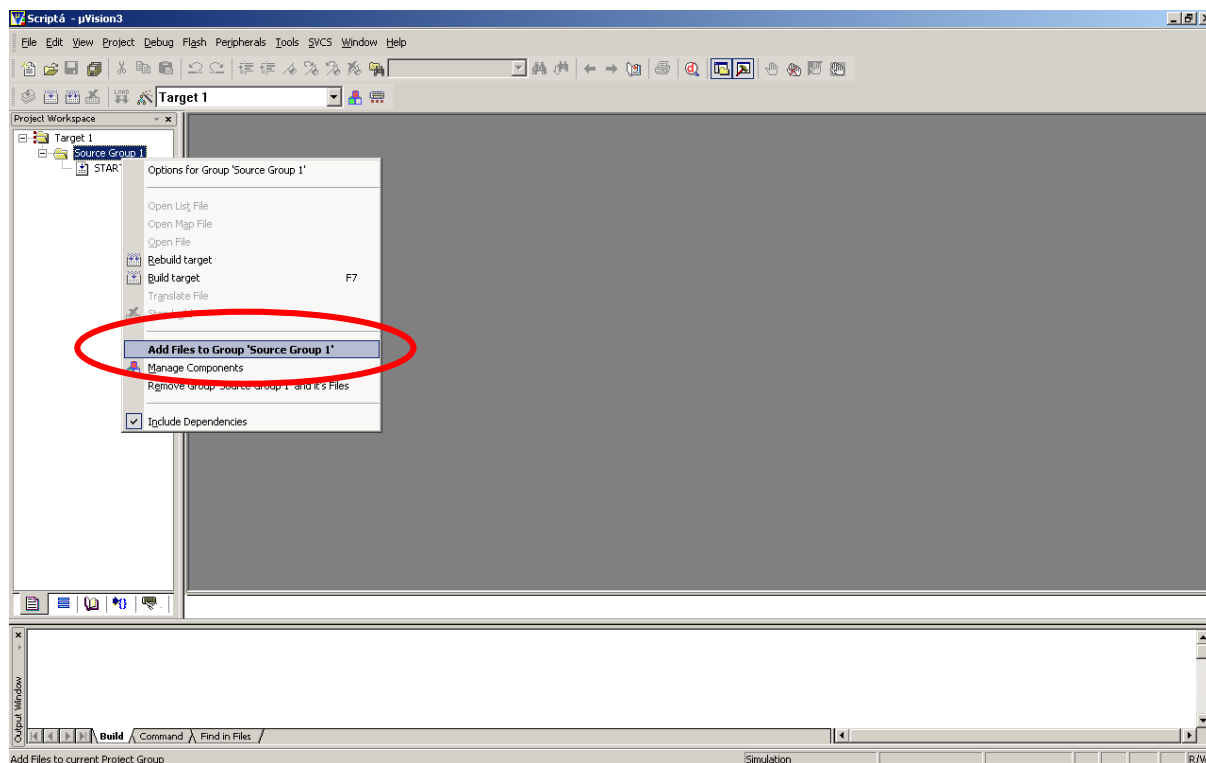
2.1. Postup pri vytváraní projektu

- vytvorenie adresára pre nový projekt v menu „Create New Project“
- voľba mikroprocesora v menu „Select Device for Target“, vid' Obrázok 23
- v okne „Project Workspace“ načítať súbory (asm, c, h, inc, lib), vid' Obrázok 22
- v okne „Options for Target“ zvoliť vlastnosti projektu, vid' Obrázok 24
- nastaviť výstupný formát preloženého súboru, Obrázok 25
- nastaviť veľkosti optimalizačného stupňa (použiteľné len pre C), Obrázok 26

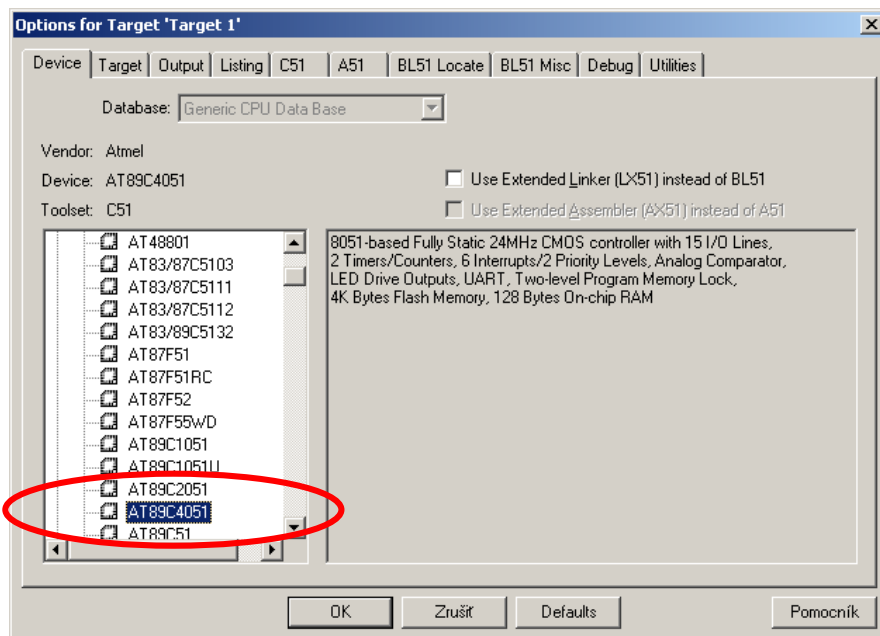
²⁴ Integrated Development Environment

2.2. Tvorba projektu krok za krokom

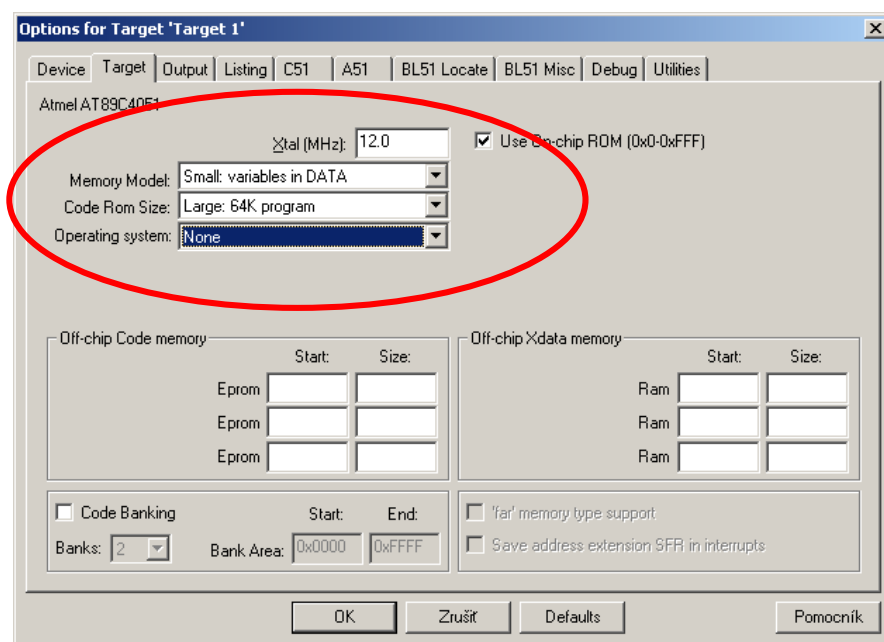
Obrázok 22, Voľba úborov pre preklad projektu v menu „Project Workspace“



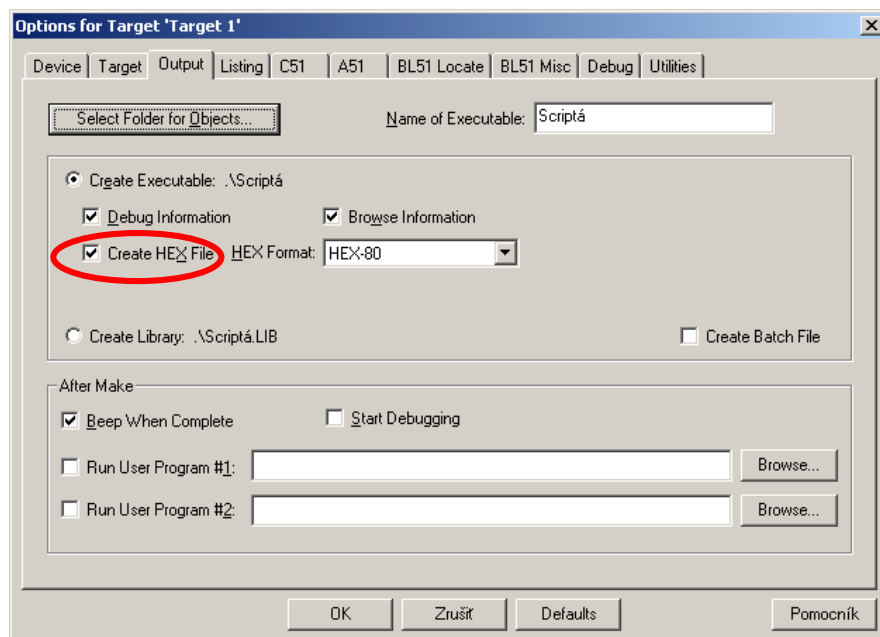
Obrázok 23, Voľba mikroprocesora v okne „Options for Target“



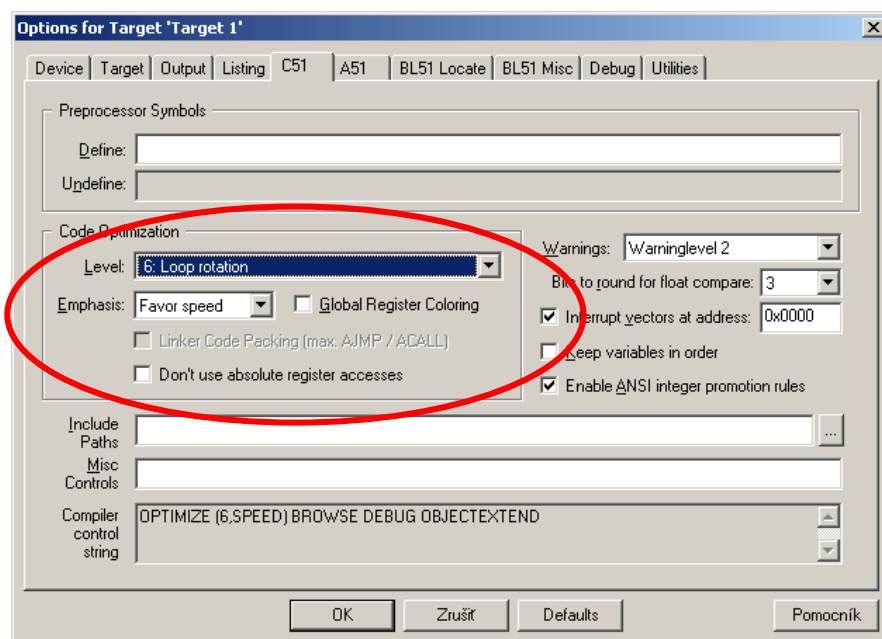
Obrázok 24, Voľba vlastností projektu v okne „Options for Target“



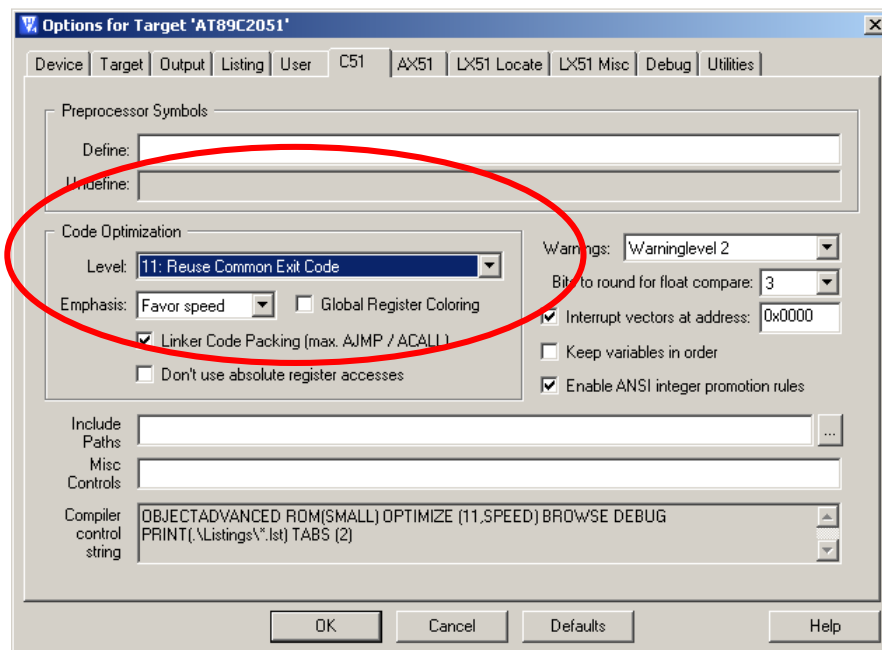
Obrázok 25, Nastavenie formátu výstupu prekladaného súboru do HEX v µVision3



Obrázok 26, Nastavenie optimalizácie a optimalizačného stupňa v μ Vision3²⁵



Obrázok 27, Použitie najvyšších optimalizačných stupňov v Ax51 a Cx51

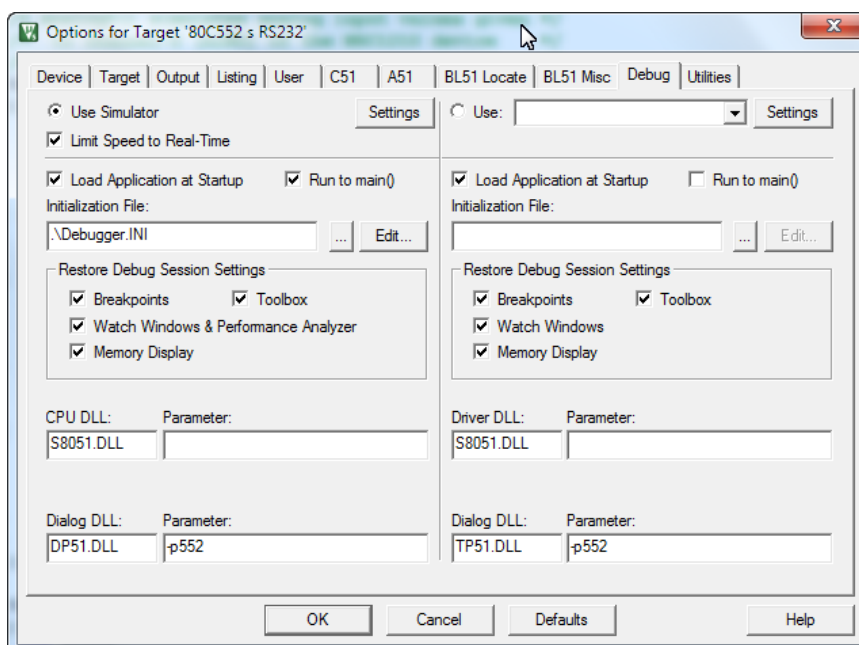


²⁵ Optimalizačný stupeň je jedným z dôležitých parametrov ovplyvňujúcim preklad výstupného programu do spustiteľného tvaru pre mikroprocesor 8051 v hex formáte. Pri použití kompilátora Ax51 a Cx51 je možné zvoliť až optimalizačný stupeň 11. Pri použití kompilátora Ax51, alebo Cx51 nie je možné ladiť aplikácie napísané s RTX51. Je to pravdepodobne preto, lebo RTX51 používa možnosti adresovať pamäť dát a programu v rozsahu 64kB, prípadne s použitím xBanking-u n*64kB. Nepodporuje 24 bitovú adresáciu používanú v modernejších a inovovaných procesoroch.

2.3. Ladenie programu pomocou integrovaného Debugger-a

Vývojové nástroje často obsahujú rôzne doplňujúce funkcie ladenia t.j. experimentálne overovanie správnej činnosti vytváraného programu. Jedným z najmocnejších nástrojov v prostredí μ Vision je integrovaný²⁶ Debugger. V súbore s príponou **.ini** napr. **debugger.ini** sú použité časti programov, procedúr, funkcií napísané v jazyku C ktoré sa súčasne vykonávajú paralelne s odlaďovaným programom. Všetky zmeny obsahov premenných²⁷, registrov sa okamžite zapisujú v prostredí a vytvárajú tak variabilne interaktívne prostredie ktoré reaguje na vstupné parametre prichádzajúce z externého programu t.j. **debugger.ini**. V tomto súbore je možné definovať pomocou množstva interných premenných prostredia Keil μ Vision ďalšie správanie sa odlaďovaného programu v závislosti na vstupných hodnotách a parametroch²⁸.

Obrázok 28, Nastavenie integrovaného Debugger-a



²⁶ Existuje možnosť použitia aj externého debugger-a, čo je v podstate špecializovaný hardware obsahujúci cieľový mikroprocesor ktorý komunikuje cez špeciálne rozhranie JTAG, pomocou ktorého je schopný čítať obsahy registrov, premenných, štruktúr pričom je možné vykonávanie programu pozastaviť, spustiť a krokovať po jednotlivých inštrukciách. Cena takéhoto hardware je pre začínajúceho programátora často veľmi vysoká a preto je lepšie keď si odlaďovanie programu skúsi ovládať cez integrovaný debugger v Keil μ Vision.

²⁷ Všetky premenné v hlavnom programe ktoré chceme použiť pri ladení v súbore **.ini** musia byť deklarované ako globálne, deklarácie funkcií majú mierne obmedzenia, preto je potrebné sledovať pri spustení **Debugger-a** okno „**Command**“, kde je vypisovaný stav úspešnosti prekladu súboru **.ini**, ktorý umožní úspešne odhaliť prípadnú chybu. Je vhodné oboznámiť sa pred použitím debuggera s internými premennými pre debugger **VTREGS** v Keil μ Vision.

²⁸ Položka **Limit Speed Real-Time** umožňuje prispôbiť rýchlosť ladeného programu na reálne hodnoty ktoré budeme dosahovať v praxi. Efekt „reálnosti“ vykonávaného programu vidíme najlepšie na výkonných počítačoch, kde by sme vykonávanie kódu bez zaškrtnutej položky nemohli kvôli rýchlosti vykonávania programu ani len sledovať.

Obrázok 29, Príklad zápisu programu pre Debugger

C:\Omega\Data\Dropbox\Záloha\C51\Sledovač fáz s prerušením\Debugger.INI

```
1  /*-----*/
2  /* Analog0() simulates analog input values given */
3  /* to channel-0 (AIN0) of the MSC1210 device */
4  /*-----*/
5  Signal void analog0 (float limit)
6  {
7      float volts,rnd;
8
9      printf ("Analog0 (%f) entered.\n", limit);
10     while (1)
11     {
12         /* forever */
13         volts = 0;
14         while (volts <= limit)
15         {
16             ain0 = volts;    /* analog input-0 */
17             swatch (0.01);   /* 0.1 Sec */
18             volts += 0.1;    /* increase voltage */
19         }
20         volts = limit;
21         while (volts >= 0.0)
22         {
23             ain0 = volts;
24             swatch (0.01);
25             volts -= 0.1;
26         }
27     }
28
29     Signal void su()
30     {
31         int Count;
32         while(1)
33         {
34             // if(++Count>=1000) P1.3=0;
35             swatch(0.00333);
36         }
37     }
38
39     Signal void Fazy_Right()
40     {
41         int Citac;
42         float TX,x;
43         while(1)
44         {
45             x+=0.00000001;
46             TX=(0.0100/3.0)+x;
47             swatch(TX);
48             P1.0=0; P1.1=1; P1.2=0;    //0x02
49             swatch(TX);
50             P1.0=0; P1.1=1; P1.2=1;    //0x06
51             swatch(TX);
52             P1.0=0; P1.1=0; P1.2=1;    //0x04
53             swatch(TX);
54             P1.0=1; P1.1=0; P1.2=1;    //0x05
55             swatch(TX);
56             P1.0=1; P1.1=0; P1.2=0;    //0x01
57             swatch(TX);
58             P1.0=1; P1.1=1; P1.2=0;    //0x03
59         }
60     }
61
62     Signal void Fazy_Left()
63     {
64         int Citac;
65         float TX,x;
66         while(1)
67         {
68             x+=0.00000001;
69             TX=(0.0100/3.0)+x;
70             swatch(TX);
71             P1.0=0; P1.2=1; P1.1=0;    //0x04
72             swatch(TX);
73             P1.0=0; P1.2=1; P1.1=1;    //0x06
74             swatch(TX);
75             P1.0=0; P1.2=0; P1.1=1;    //0x02
```

C:\Omega\Data\Dropbox\Záloha\C51\Sledovač fáz s prerušením\Debugger.INI

```
76     swatch(TX);
77     P1.0=1; P1.2=0; P1.1=1;      //0x03
78     swatch(TX);
79     P1.0=1; P1.2=0; P1.1=0;      //0x01
80     swatch(TX);
81     P1.0=1; P1.2=1; P1.1=0;      //0x05
82 }
83 }
84
85 // Create signal functions for buttons
86 signal void pb1 (void)
87 {
88     P3^=0x10;
89     swatch (2);
90 }
91
92 signal void pb2 (void)
93 {
94     P1^=0x10;
95     swatch (2);
96     P1^=0x10;
97 }
98
99 signal void pb3 (void)
100 {
101     P4^=0x80;
102     swatch (2);
103     P4^=0x80;
104 }
105
106 // Create signal functions for car detectors
107 signal void cd1 (void)
108 {
109     P3^=0x04;
110     swatch (2);
111     P3^=0x04;
112 }
113
114 signal void cd2 (void)
115 {
116     P3^=0x08;
117     swatch (2);
118     P3^=0x08;
119 }
120
121 // Assign toolbox buttons to signal functions
122 define button "ZSF", "pb1()"
123 define button "OP", "pb2()"
124 define button "Zavesenie frekvencie", "pb3()"
125 define button "Nadprud slabý", "cd1()"
126 define button "Nadprud silný", "cd2()"
127
128 // start volatage ramp on AIN0
129 analog0 (4.9)
130 //su()
131 Fazy_Right()
132 //Fazy_Left()
133 //g,main
134
135 // fixed voltage on AIN0 .. AIN7
136 AIN0=0.000000
137 AIN1=0.000000
138 AIN2=2.500000
139 AIN3=0.000000
140 AIN4=0.000000
141 AIN5=0.000000
142 AIN6=0.000000
143 AIN7=0.000000
```

3 Kompilátor C pre architektúru 8051

3.1. Minimálne požiadavky na prekladač

- Inteligentné rozmiestňovanie kódu programu a dát,
- Možnosť použitia dátových typov (bit, byte, word, dword, longint, real ...),
- Špeciálne metódy adresovania (xBanking, 24 bit adresácia),
- Rôzne pamäťové modely (SMALL, COMPACT, LARGE, HUGE),
- Riadenie aplikácií na úrovni prerušení.

Množstvo aplikácií je napísaných vo vyšších programovacích jazykoch C, EC++, Pascal, Basic, Fortran a iných. Po podrobnejšom preskúmaní systémových zdrojov jednotlivých mikroprocesorov architektúry 8051 je možné povedať, že ak samotná systémová výbava architektúry procesora je obmedzenejšia v konečnom dôsledku sa znižuje efektívnosť samotného generovaného programovaného kódu prekladačom.

U niektorých architektúr mikroprocesorov, napríklad AVR je zaistená optimálna veľkosť výsledného programového kódu nielen pomocou hardwaru, ale aj vďaka použitej architektúre ktorá obsahuje 32 univerzálnych registrov (GPR – General Purpose Register). Vďaka tejto 16 bitovej architektúre je už možné použiť ľubovoľný register pre aritmetickú, logickú alebo vo funkcii ukazovateľa dát a v spojení s jednostupňovým paralelným spracovaním inštrukcií (pipeling) je možné v jedinom hodinovom cykle spracovať viacero inštrukcií.

3.2. Moderné požiadavky na prekladač

Ideálny kompilátor – prekladač by mal teda vytvoriť cieľový program tak dobre ako keby bol napísaný v assembler-i. Vďaka neustálemu vývoju množstva existujúcich optimalizačných metód prekladačov sa k tomuto cieľu blížíme. Pri účinnom využití architektúry mikroprocesora je možné nadbytočnosť (overhead) kódu znížiť o cca. 30% v porovnaní s programom, ktorý je napísaný v assembler-i. V prípade že je algoritmus napísaný programátorom neefektívne, nepomôže ani špičkový kompilátor s optimalizačnými modulmi.

Až pri nových architektúrach AVR je braný do úvahy ohľad na **HLL** (High Level Language), kde dochádza k zníženiu prebytočného programového kódu vygenerovanému v HLL takmer na nulu. Tento proces je ale závislý hlavne na programátorovi ktorý musí dodržiavať pomerne veľké množstvo pravidiel pri písaní programu, aby vytvoril výsledný vysoko efektívny programový kód.

4 Tvorba programov v jazyku C51

4.1. Prehľad používaných architektúr mikroprocesorov

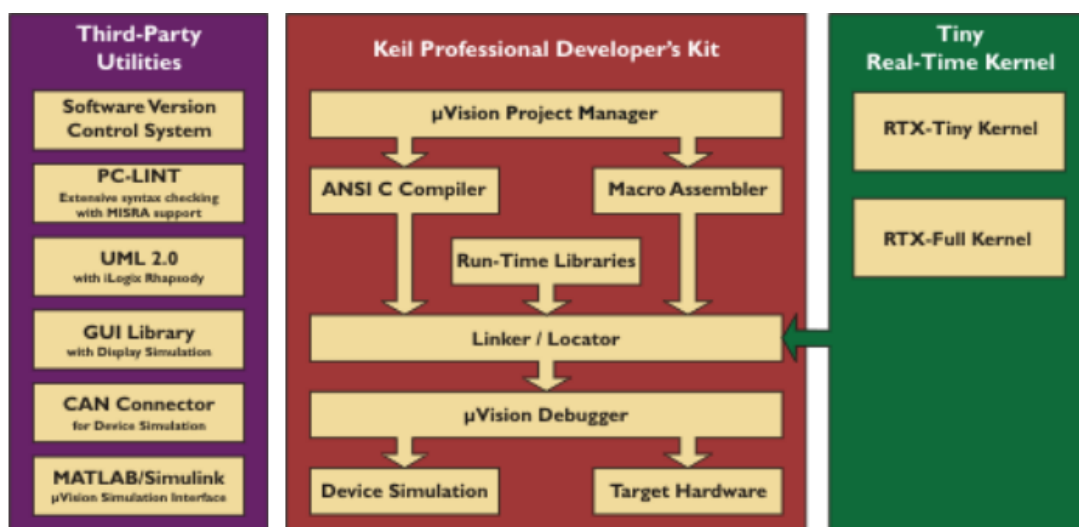
- Harvard,
- Von Neuman,
- Hybridná (*Harvard + Von Neuman*),
- CISC (*Complex Instruction Set Computer*),
- RISC (*Reduced Instruction Set Computer*),
- CISP (*Configurable Instruction Set Computer*).

4.2. Prekladače jazyka C pre mikroprocesory

Hlavnou úlohou prekladača jazyka C je:

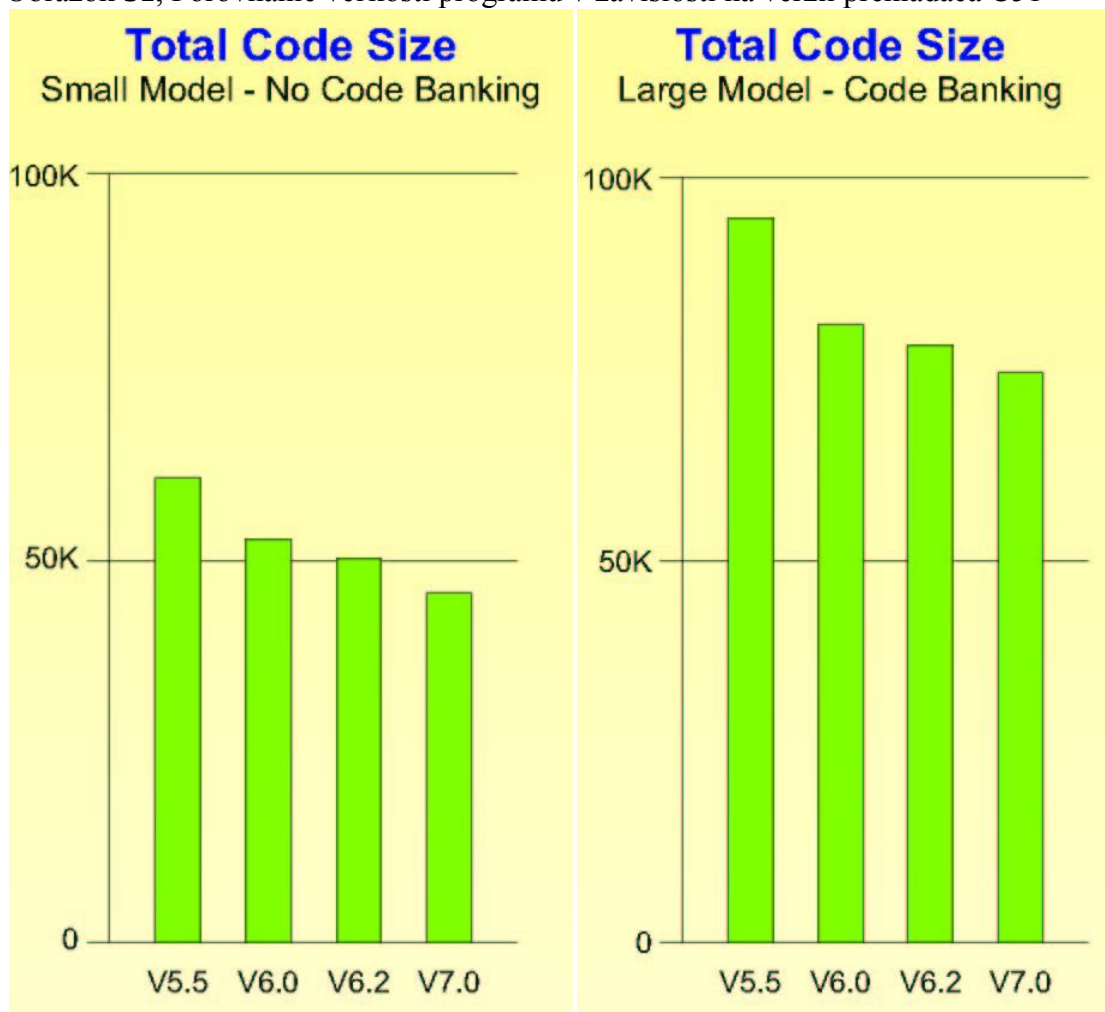
- Automaticky rozmiestňovať programový kód a dát,
- Mať otvorenú architektúru umožňujúcu využitie častí programového kódu, alebo knižníc napísaných v inom jazyku (*BASIC, PASCAL, C, FORTRAN, ASM ...*),
- Umožňovať rôzne metódy adresovania pre iné CPU,
- Používať rôzne pamäťové modely²⁹ (*SMALL, COMPACT, LARGE ...*), a špeciálne údajové typy (*char, int, float, double, string, pointer ...*),
- Riadiť program na úrovni prerušenia.

Obrázok 30, Programovací model vývojového nástroja μ Vision4 a vyšší



²⁹ Vyššie programovacie jazyky generujú síce univerzálny programový kód, ale pri porovnaní s assembler-om je značne väčší „nafúknutý“ a samozrejme menej efektívny.

Obrázok 31, Porovnanie veľkostí programu v závislosti na verzii prekladača C51



Veľkosť pamäťového priestoru použitá prekladačom C v jednotlivých pamäťových modeloch nie je pre programátora príliš podstatná, pretože je to pamäťový priestor nevyhnutne potrebný pre činnosť programu v závislosti na množstve použitých premenných. V prípade nedostatku pamäťového priestoru prekladač ohlásí tento nedostatok pamäti programátorovi. Len v jednom prípade je potrebné dávať pozor na maximálne množstvo pridelenej pamäti a to vtedy ak používame reentrant³⁰ funkcie, pretože funkcie ktoré sú reentrant-né používajú pamäť IDATA³¹ mikroprocesora pre svoju činnosť. Takto sa môže veľmi ľahko stať, že niekoľkonásobné vyvolanie funkcie môže mať za následok vyčerpania prideleného pamäťového priestoru a tým pádom programu. Konkrétne sa jedná o pamäťový model COMPACT, ktorý sa tak často nepoužíva ako model, prípadne LARGE³².

³⁰ Reentrantné funkcie používajú vlastný „environment“ t.j. prostredie alokujúce oddelený pamäťový priestor pre každú súčasne viacnásobne volanú funkciu z hlavného programu, alebo z prerušenia.

³¹ Oblasť pamäti RAM mikroprocesora 8051 v oblasti adres od 080h do 0FFh, ktorá je nepriamo adresovateľná pomocou registrov R0 a R1, napr. `mov a,@R0`.

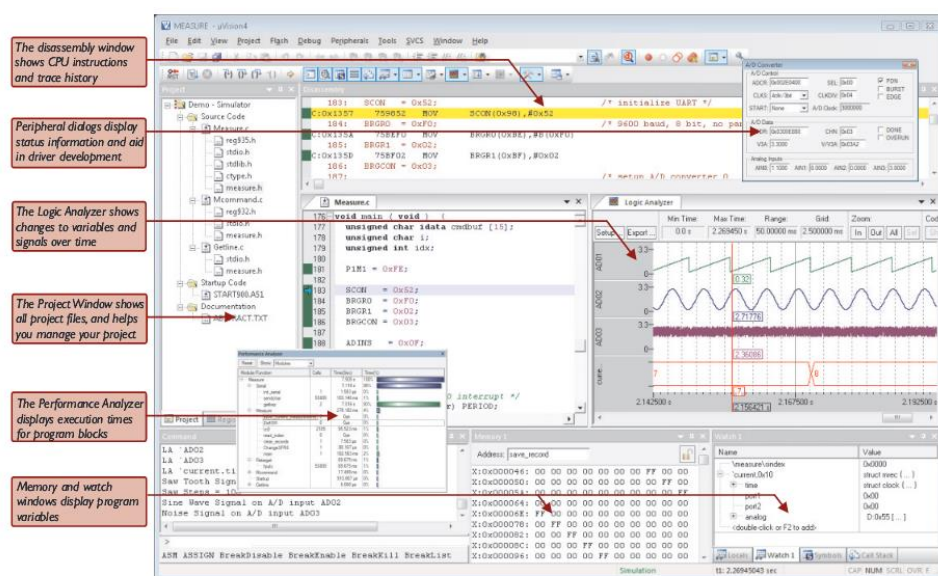
³² Využíva oblasť pamäti RAM od 00h do 0FFh t.j. DATA+IDATA a 0000h do 0FFFFh t.j. XRAM, všetko v rozsahu 0 až 64kB.

Operačný systém RTX51 Full vyžaduje nevyhnutne k svojej činnosti navyše externú pamäť dát XRAM o veľkosti minimálne 1kB plus pamäť pre použité premenné a úlohy. Pre RTX51 Tiny nie je vyžadovaná externá pamäť, stačí pamäť³³ RAM o veľkosti 256 Byte.

Obrázok 32, Vývoj aplikácie s Cx51 a RTX51



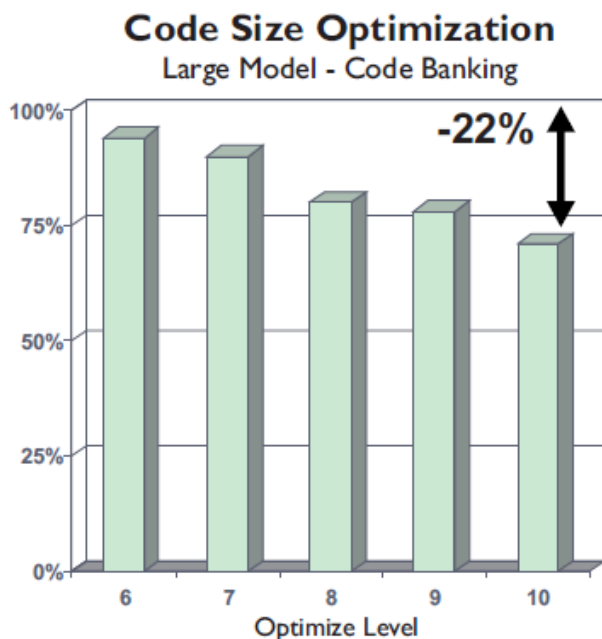
Obrázok 33, Vývoj aplikácie v prostredí μVision4



The μVision development platform is easy to use and it helps you to quickly create embedded programs that work. The μVision editor and debugger are integrated in a single application that provides a seamless embedded project development environment for editing, simulating, Flash programming and testing in target hardware.

³³ Nastavenie je závislé na hodnote **RAMTOP** v súbore **conf_tiny.a51**, kde hodnota **07Fh** je použiteľná pre mikroprocesory osadených maximálne 128 Byte internej pamäti DATA v rozsahu 00h až 07Fh. RTX51 Tiny správne funguje pokiaľ nedosiahne 90% obsadenosti pamäti DATA, IDATA t.j. RAM. V opačnom prípade to vedie k nekorektnej funkcii niektorých taskov, čo je pomerne veľmi ťažko v praxi identifikovať. V prípade potreby väčšieho množstva pamäti pre program je možné hodnotu **STACK** zmeniť na hodnotu 00h, čím zrušíme pamäť pre zásobník a aplikácie dostanú k dispozícii uvoľnenú pamäť.

Obrázok 34, Optimalizácia programového kódu kompilátorom Cx51



Obrázok 35, Optimalizačné stupne µVision

Level	Description
0	Constant Folding: The compiler performs calculations that reduce expressions to numeric constants, where possible. This includes calculations of run-time addresses. Simple Access Optimizing: The compiler optimizes access of internal data and bit addresses in the 8051 system. Jump Optimizing: The compiler always extends jumps to the final target. Jumps to jumps are eliminated.
1	Dead Code Elimination: Unused code fragments and artifacts are eliminated. Jump Negation: Conditional jumps are closely examined to see if they can be streamlined or eliminated by the inversion of the test logic.
2	Data Overlaying: Data and bit segments suitable for static overlay are identified and internally marked. The BL51 Linker/Locator has the capability, through global data flow analysis, of selecting segments which can then be overlaid.
3	Peephole Optimizing: Redundant MOV instructions are removed. This includes unnecessary loading of objects from the memory as well as load operations with constants. Complex operations are replaced by simple operations when memory space or execution time can be saved.
4	Register Variables: Automatic variables and function arguments are located in registers when possible. Reservation of data memory for these variables is omitted. Extended Access Optimizing: Variables from the IDATA, XDATA, PDATA and CODE areas are directly included in operations. Intermediate registers are frequently unnecessary. Local Common Subexpression Elimination: The compiler detects multiple uses of the same expression or subexpression. The result of the first expression is saved and reused when possible. Superfluous expression calculations are eliminated from the code. Case/Switch Optimizing: Code involving switch and case statements is optimized using jump tables or jump strings. Simple Loop Optimizing: Program loops that fill a memory range with a constant are converted and optimized.
5	Global Common Subexpression Elimination: Identical subexpressions within a function are calculated only once when possible. The intermediate result is stored in a register and reused.
6	Loop Rotation: Program loops are rotated if the resulting program code is faster and more efficient. The loop expression of for and while loops is evaluated once at the top of the loop and then at the bottom of the loop. This optimization generates more code but speeds up execution.
7	Extended Index Access Optimizing: DPTR is used for register variables where appropriate. Pointer and array access are optimized for both execution speed and code size.
8	Common Tail Merging: When there are multiple calls to a single function, some of the setup code can be reused, thereby reducing program size.
9	Common Block Subroutines: Recurring instruction sequences are detected and converted into subroutines. The Cx51 Compiler rearranges code to obtain larger recurring sequences.
10	Rearrange Code (Linker Optimization): When detecting common block subroutines, code is rearranged to obtain larger recurring sequences.
11	Reuse of Common Exit Code (Linker Optimization): Identical exit sequences are reused. This may reduce the size of common block subroutines even further. This optimization level generates the most compact program code possible.

4.3. Príklad zápisu programu v C51

Príklad: Napíšte program, ktorý vyšle na sériové rozhranie textový reťazec "Hello world !" s prenosovou rýchlosťou 1200b.s^{-1} , 1 štart a stop bit, 8 bit slovo, bez parity s 16MHz kryštálom.

Príklad zápisu programu v jazyku C51:

c:\Keil\C51\Examples\HELLO\HELLO.C

```
1  /*-----*/
2  HELLO.C
3
4  Copyright 1995-2005 Keil Software, Inc.
5  /*-----*/
6
7  #include <REG52.H>          /* special function register declarations */
8                             /* for the intended 8051 derivative */
9
10 #include <stdio.h>          /* prototype declarations for I/O functions */
11
12
13 #ifdef MONITOR51            /* Debugging with Monitor-51 needs */
14 char code reserve [3] _at_ 0x23; /* space for serial interrupt if */
15 #endif                      /* Stop Execution with Serial Intr. */
16                             /* is enabled */
17
18
19 /*-----*/
20 The main C function. Program execution starts
21 here after stack initialization.
22 /*-----*/
23 void main (void) {
24
25 /*-----*/
26 Setup the serial port for 1200 baud at 16MHz.
27 /*-----*/
28 #ifndef MONITOR51
29     SCON = 0x50;             /* SCON: mode 1, 8-bit UART, enable rcvr */
30     TMOD |= 0x20;            /* TMOD: timer 1, mode 2, 8-bit reload */
31     TH1 = 221;               /* TH1: reload value for 1200 baud @ 16MHz */
32     TR1 = 1;                 /* TR1: timer 1 run */
33     TI = 1;                  /* TI: set TI to send first char of UART */
34 #endif
35
36 /*-----*/
37 Note that an embedded program never exits (because
38 there is no operating system to return to). It
39 must loop and execute forever.
40 /*-----*/
41 while (1)
42 {
43     P1 ^= 0x01;              /* Toggle P1.0 each time we print */
44     printf ("Hello World\n"); /* Print "Hello World" */
45 }
46
47
48
49
```

Výpočet hodnoty TH1 pre časovač v režime 1 pre vysoké a nízke rýchlosti:

$$b^{-s} = \frac{2^{SMOD}}{32} \cdot \frac{f_{osc}}{12 \cdot (256 - TH1)}$$

$$b^{-s} = \frac{2^{SMOD}}{32} \cdot \frac{f_{osc}}{12 \cdot (65536 - [TH1, TL1])}$$

$$TH1 = 256 - \frac{2^{SMOD} \cdot f_{osc}}{384 \cdot b^{-s}} \quad (5.)$$

4.4. Deklarácia zložitejších pamäťových štruktúr v C (statická a dynamická alokácia)

Pod pojmom zložitejšie pamäťové štruktúry rozumieme deklarovanie viacprvkovej heterogénnej množiny rôznych typov premenných, pričom sú zvonku viditeľné ako jednoliaty samostatný celok.

Obrázok 36, Deklarácia pamäťovej štruktúry

```
typedef xdata struct
{
    unsigned char Preamble,Dst,Src;
    unsigned int Size;
    unsigned char Data[SizeOfPacketData];
    unsigned char Crc;
}
Packet51;
```

Pri niektorých prekladačoch³⁴ sa môžeme stretnúť so štruktúrami s atribútom určujúcim veľkosť premennej v bitoch.

Obrázok 37, Deklarácia pamäťovej štruktúry s určením veľkosti

```
typedef xdata struct
{
    unsigned int z:10;           //Len 10 bitov
    unsigned int x:8;            //Len 8 bitov
    unsigned char y:2;           //Len 2 bity
}
ADC_Result;
```

4.4.1. Statická alokácia premenných v jazyku C

Priradenie štruktúry „Packet51“ premennej „packet“ zabezpečíme v jazyku C pomocou nasledujúceho zápisu:

```
xdata Packet51 packet;
```

4.4.2. Dynamická alokácia premenných v jazyku C

Dynamické alokovanie pamäti sa v jazyku C používa v prípadoch, keď je potrebné dynamicky pridelovať podľa skutočných potrieb množstvo pamäti, ktorá nie je známa.

³⁴ MDK-ARM umožňuje použiť štruktúru s atribútom **packed**. Umožňujúcim do jednej premennej uložiť väčší počet menších premenných ktorých spoločná veľkosť je rovná súčtu použitej premennej.

Na zabezpečenie dostatočného množstva pamäte (alokovanie) sa v programovacom jazyku C používa funkcia ***malloc_mempool[]*** a funkcia ***init_mempool()*** s funkciou ***malloc()***³⁵, ktorá inicializuje množstvo pamäti, alebo alokovanú pamäť pre potreby programátora.

³⁵ V prípade nedostatku pamäte pre alokovanie premennej je funkciou ***malloc()*** vrátená hodnota ***NULL***.

V prípade dynamickej alokácie premennej „Recv_Message“ štruktúry „Packet51“ je potrebné použiť nižšie uvedený postup:

- a. Vytvoriť v pamäti **xdata**³⁶ dostatočný priestor pre dynamicky pridelenú pamäť:
unsigned char xdata malloc_mempool[2(sizeof(Packet51)+5)];*
- b. inicializácia celého pamäťového bloku pamäti **xdata** pomocou funkcie jazyka C:
init_mempool(&malloc_mempool,sizeof(malloc_mempool));
- c. priradenie statickej štruktúry „Packet51“ premennej typu ukazovateľ na štruktúru s názvom „Recv_Message“:
*xdata Packet51 *Recv_Message;*
- d. inicializácia premennej „Recv_Message“, adresy prideleného pamäťového priestoru:
Recv_Message=(Packet51)malloc(sizeof(Packet51));*
- e. naplnenie jednotlivých položiek štruktúry údajmi:
Recv_Message->Preamble='^';
Recv_Message->Dst=0xAB;
Recv_Message->Dst=0xCD;
...

³⁶ Programátor musí vytvoriť potrebný priestor v dynamickej pamäti pre údaje a aj pre tzv. popisovač bloku, ktorého veľkosť môže závisieť od pamäťového modelu mikroprocesora. Všetka dostupná pamäť musí byť vytvorená v externej pamäti programu. Programátor musí zabezpečiť **init.a51** do projektu ktorý obsahuje kompletný pamäťový management dynamicky alokovanej pamäte.

4.5. Deklarácia zložitejších pamäťových štruktúr v jazyku C pomocou funkcií RTX51

Pri programovaní aplikácií pre mikroprocesory sa niekedy nevyhneme použitiu zložitejších pamäťových premenných, alebo pamäťových štruktúr. Ich použitie umožní programátorovi lepšie a efektívnejšie vytvárať časti programového kódu a jednoduchšie pristupovať k obsahom premenných. Jednoduché premenné jazyka C a ich deklarácia sú uvedené v každej učebnici jazyka C napr. (Keringham, a iní, 1998).

Obrázok 38, Deklarácia jednoduchšej štruktúry v C

```
typedef xdata struct //Musí byť deklarovaná a používaná ako je to uvedené v tomto tvare, inak správne nefunguje !!!
{
    unsigned char Preamble,Dst,Src;
    unsigned int Size;
    unsigned char Data[SizeOfPacketData];
    unsigned char Crc;
}
Packet51;

xdata Packet51 *ReceivedMessage;
xdata Packet51 Packet = {Token,1,2,sizeof(Packet.Data), "!!! Diversant Software !!!", NULL};
xdata unsigned int *MessageToSend, *ReceivedToBuffer, *MTS;

xdata Packet51 AAA;

void System(void) _task_ SYSTEM _priority_ 0
{
    signed char RtxCompletion;
    unsigned char fcs;
    unsigned int *ptr;
    RtxCompletion=os_set_slice(15360);
    RtxCompletion=os_create_pool(SizeMemBlock,MemoryPool,SizeMemPool); //Vytvorim MemoryPool v pamäti XRAM pre datové štruktúry
    RtxCompletion=os_create_task(SEND);
    while(1)
    {
        AAA=Packet;
        Vytvor_Packet((char)rand(), (char)rand(), "!!! Diversant Software !!!", &AAA);
    }
}
```

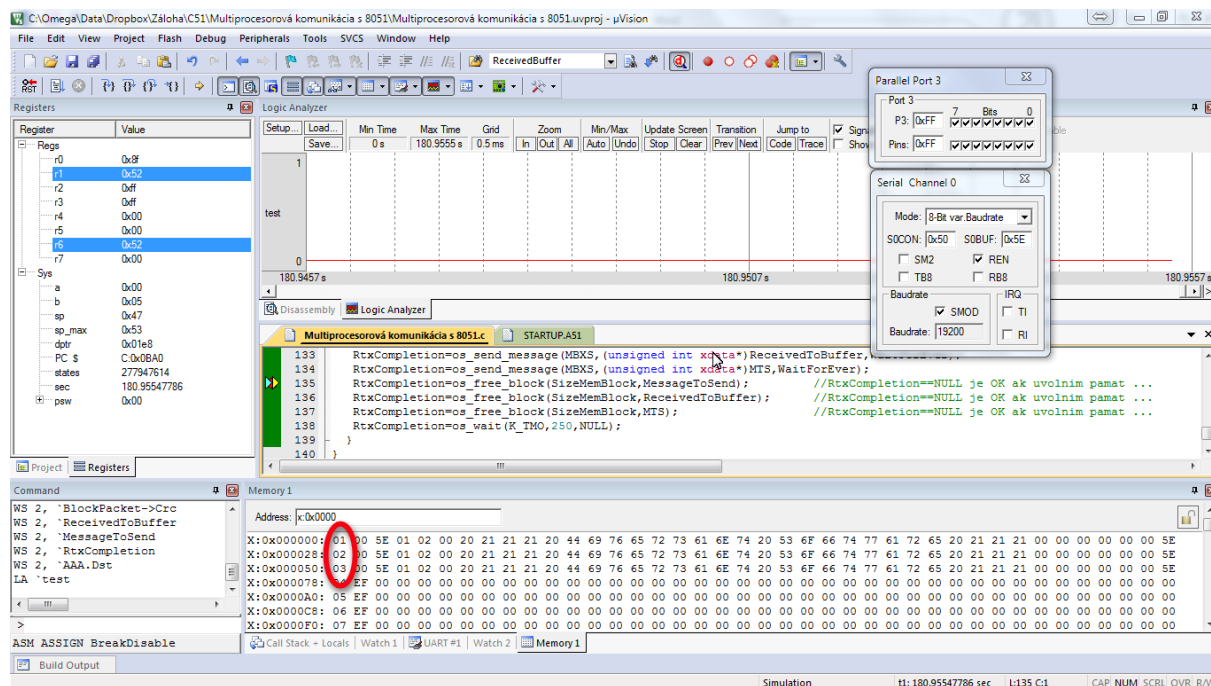
Obrázok 38 názorne ukazuje vytvorenie štruktúry **Packet51** ako pole rôznych údajových typov a jej následné priradenie pre premennú³⁷ **Packet** a **AAA**. Tieto dve premenné sú absolútne rovnocenné. S výhodou je ich použiť na prvotnú inicializáciu napr. **AAA=Packet**, kde sa štruktúra **AAA** naplní obsahom štruktúry **Packet** bez použitia funkcie **memcpy()**. Prístup do štruktúry je možné zápisu **Packet.Dst**, prípadne pre prístup do poľa **Packet.Data[i]** použije tento typ zápisu. V prípade ak poznáme len adresu pamäťovej premennej odoslanej mailbox-om ako údajový typ integer (**unsigned int xdata***)&**ReceivedMessage**) môžeme použiť zápis pre prístup do premennej **ReceivedMessage->Dst**, alebo **ReceivedMessage->Data[i]**. Premenná **ReceivedMessage** musí ukazovať svojím obsahom na adresu pamäťového bloku³⁸

³⁷ Takto definovaná premenná je umiestnená prekladačom staticky do pamäti **xdata** a nie je vôbec potrebné programátorom rezervovať miesto pre danú pamäťovú štruktúru.

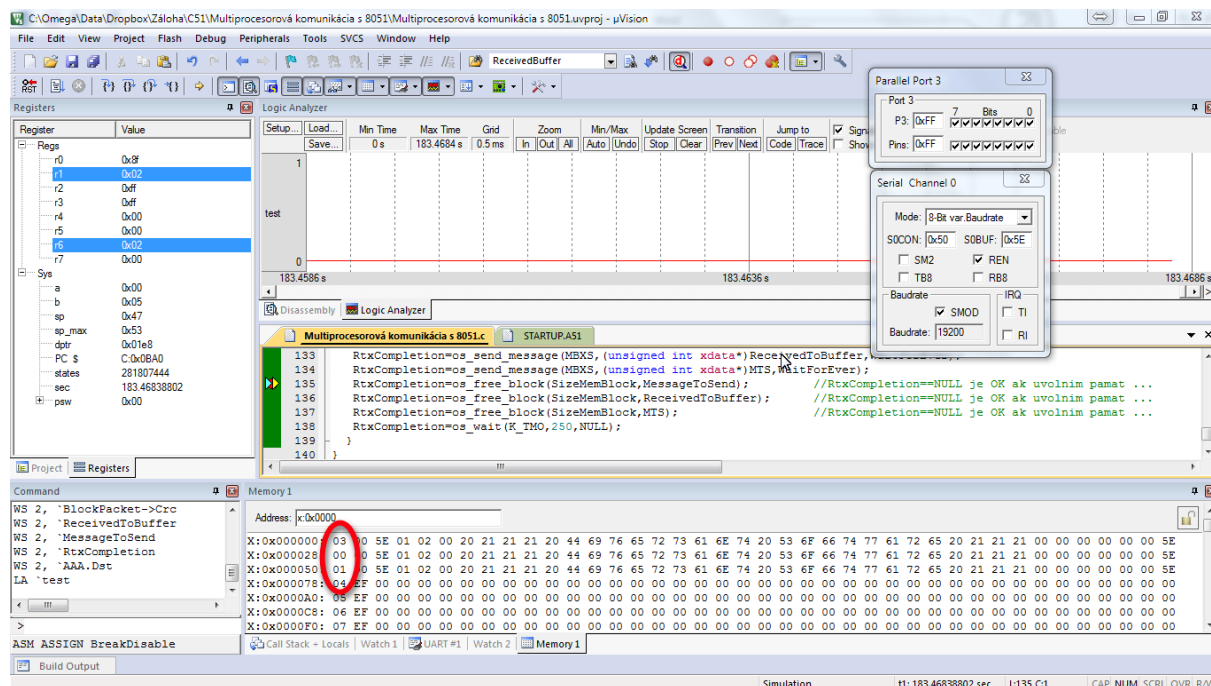
³⁸ Pri alokovaní pamäťových blokov pomocou **MessageToSend=os_get_block(SizeMemBlock)** je potrebné postupovať rovnako ako pri plnení zásobníka procesoru pomocou inštrukcií 8051 PUSH a POP. Pri nedodržaní tohto postupu pri odladovaní programu je vidieť v header-i hlavičky alokovaného bloku pamäti po použití **RtxCompletion=os_free_block(SizeMemBlock,MessageToSend)** vo výpise pamäti nesprávne priradovanie poradových čísel alokovaných blokov pamäti.

ktorý je rezervovaný pomocou funkcie **MessageToSend=os_get_block(SizeMemBlock)** a po ukončení práce so štruktúrou je potrebné rezervovanú pamäť uvoľniť pomocou funkcie **os_free_block(SizeMemBlock,MessageToSend)**. Premenná **Packet** môže po vytvorení obsahovať rôzne a náhodné hodnoty, preto musíme jej obsah inicializovať funkciou **memset(&Packet,0x00,sizeof(Packet))**.

Obrázok 39, Správne uvoľnenie pamäti pomocou os_free_block()



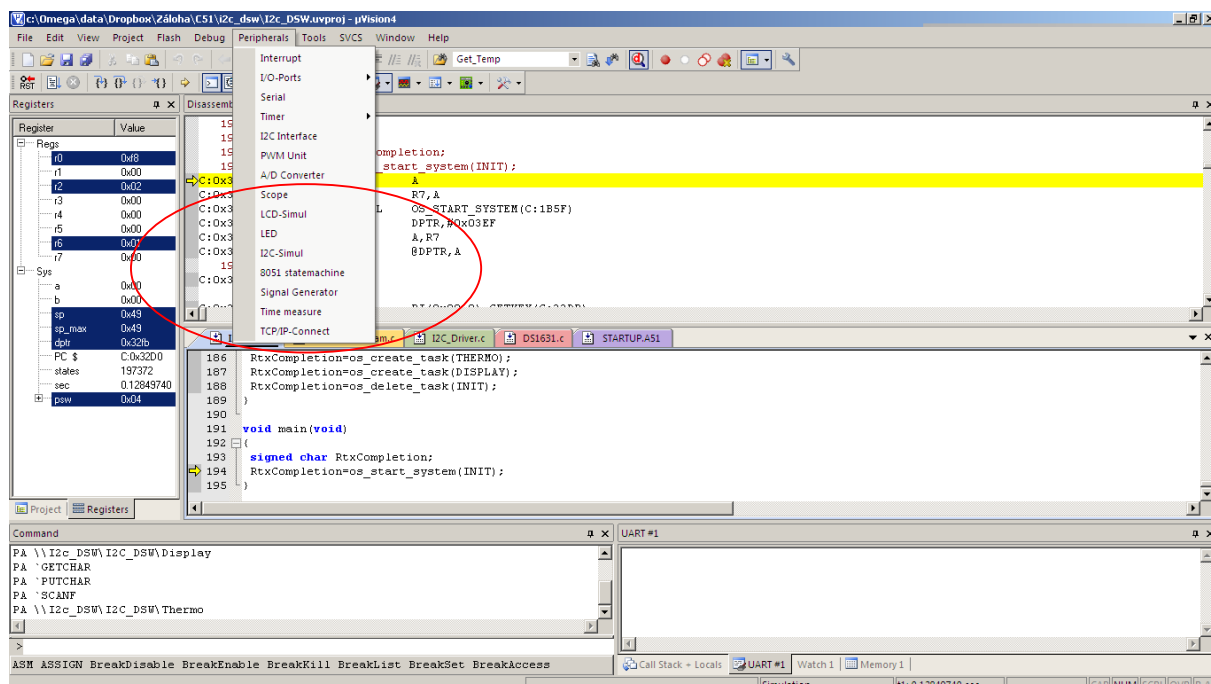
Obrázok 40, Nesprávne uvoľnenie pamäti pomocou os_free_block()



4.6. Ladenie – simulácia aplikácií v µVision

Moderné mikroprocesory na svojom jadre obsahujú množstvo periférnych obvodov, ktoré programátor len s veľkými ťažkosťami dokáže odladiť – odsimulovať na PC a v konkrétnej aplikácii. Na zjednodušenie tejto náročnej činnosti bolo programátormi tretích strán naprogramovaných množstvo aplikácií³⁹ ktoré umožňovali jednoduchým spôsobom simulovať funkciu niektorých obvodov mikroprocesora priamo v prostredí µVision.

Obrázok 41, Modifikované prostredie µVision4 pre ladenie aplikácií s AGSI⁴⁰ ovládačmi



Pre správnu činnosť je potrebné modifikovať priečinok **C:\KEIL** kde je uložený súbor **tools.ini** s inicializačnými rutinami potrebných pre začlenenie AGSI ovládačov do systému.

Inštalácia AGSI ovládačov v TOOLS.INI

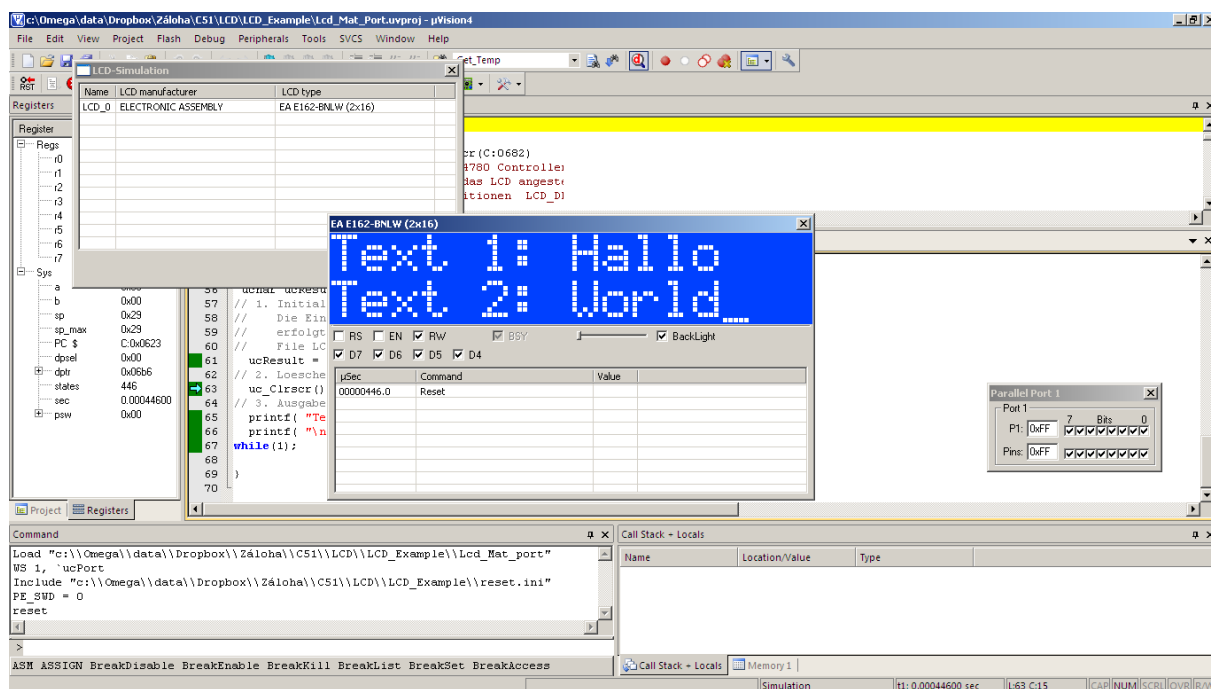
[C51]

```
PATH="C:\Keil\C51\"
VERSION=V9.54a
AGSI1=SCOPE.DLL ("Scope simulation")
AGSI2=LCD.DLL ("LCD simulation")
AGSI2=LED.DLL ("LED simulation")
AGSI3=I2C.DLL ("I2C simulation")
AGSI4=Statemachine.dll ("8051 statemachine")
AGSI5=Signalgenerator.dll ("Signal generator")
AGSI6=Timemeasure.dll ("Timemeasure")
AGSI7=TCPIP.dll ("TCPIP connection")
```

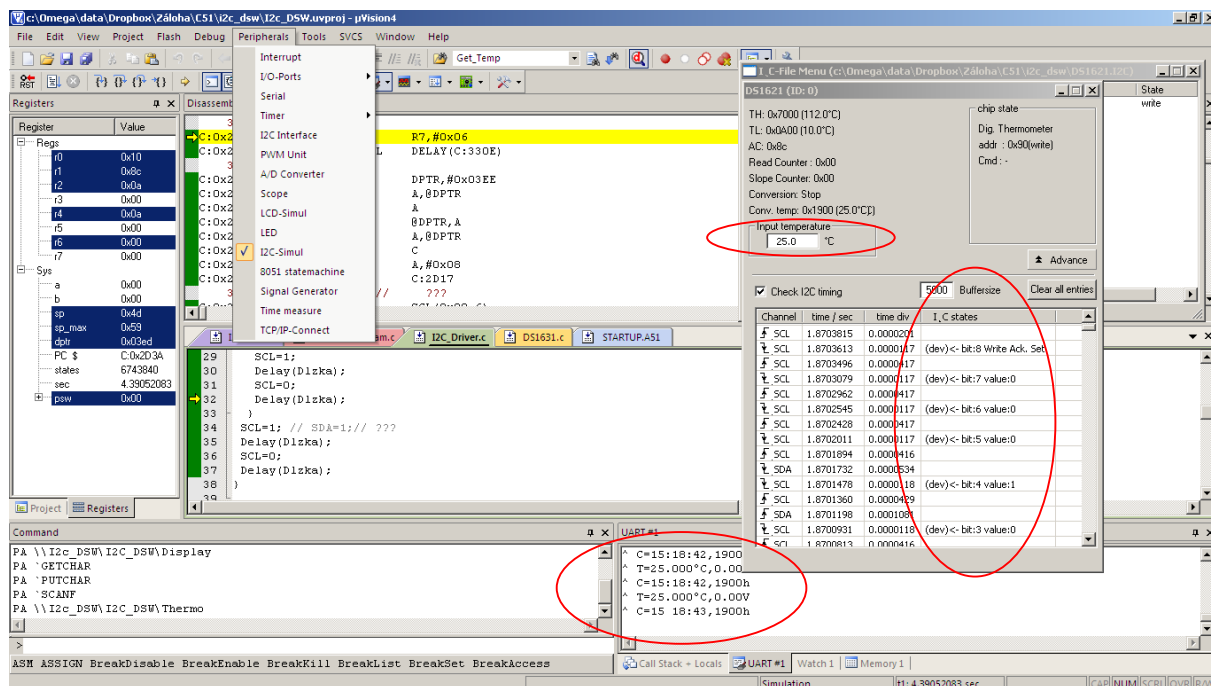
³⁹ AGSI ovládače sú dostupné zdarma na <http://www.c51.de/c51.de/Dateien/uVision2DLLs.php?Spr=EN>

⁴⁰ Bližšie informácie na http://www.keil.com/uvision/db_sim_agssi.asp

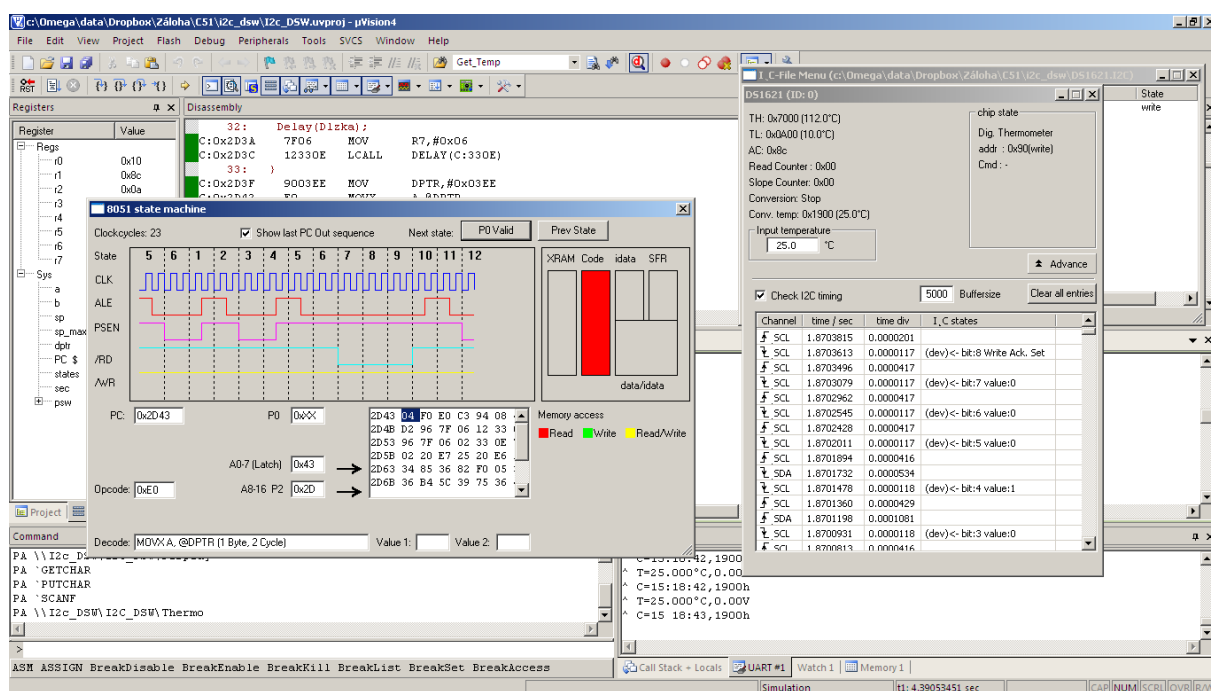
Obrázok 42, Simulácia LCD display-a AGSI ovládačmi



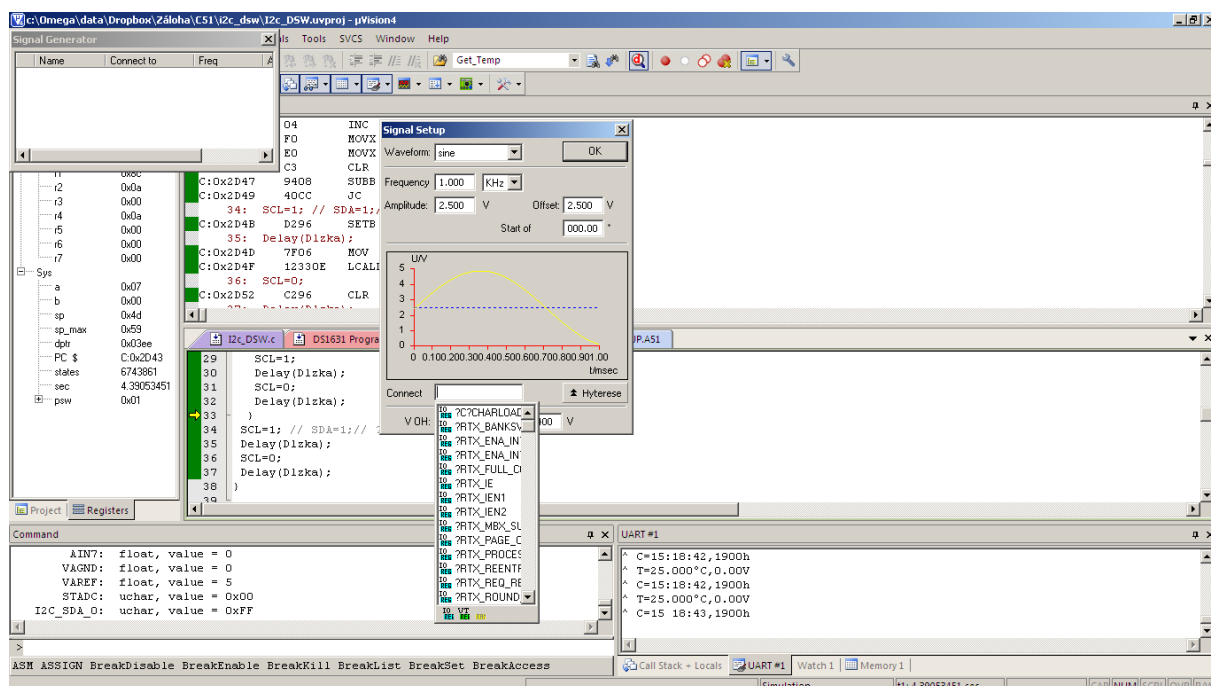
Obrázok 43, Simulácia I²C s AGSI ovládačmi



Obrázok 44, Simulácia stavu procesora s AGSI ovládačmi



Obrázok 45, Simulácia generátora priebehov s AGSI ovládačmi



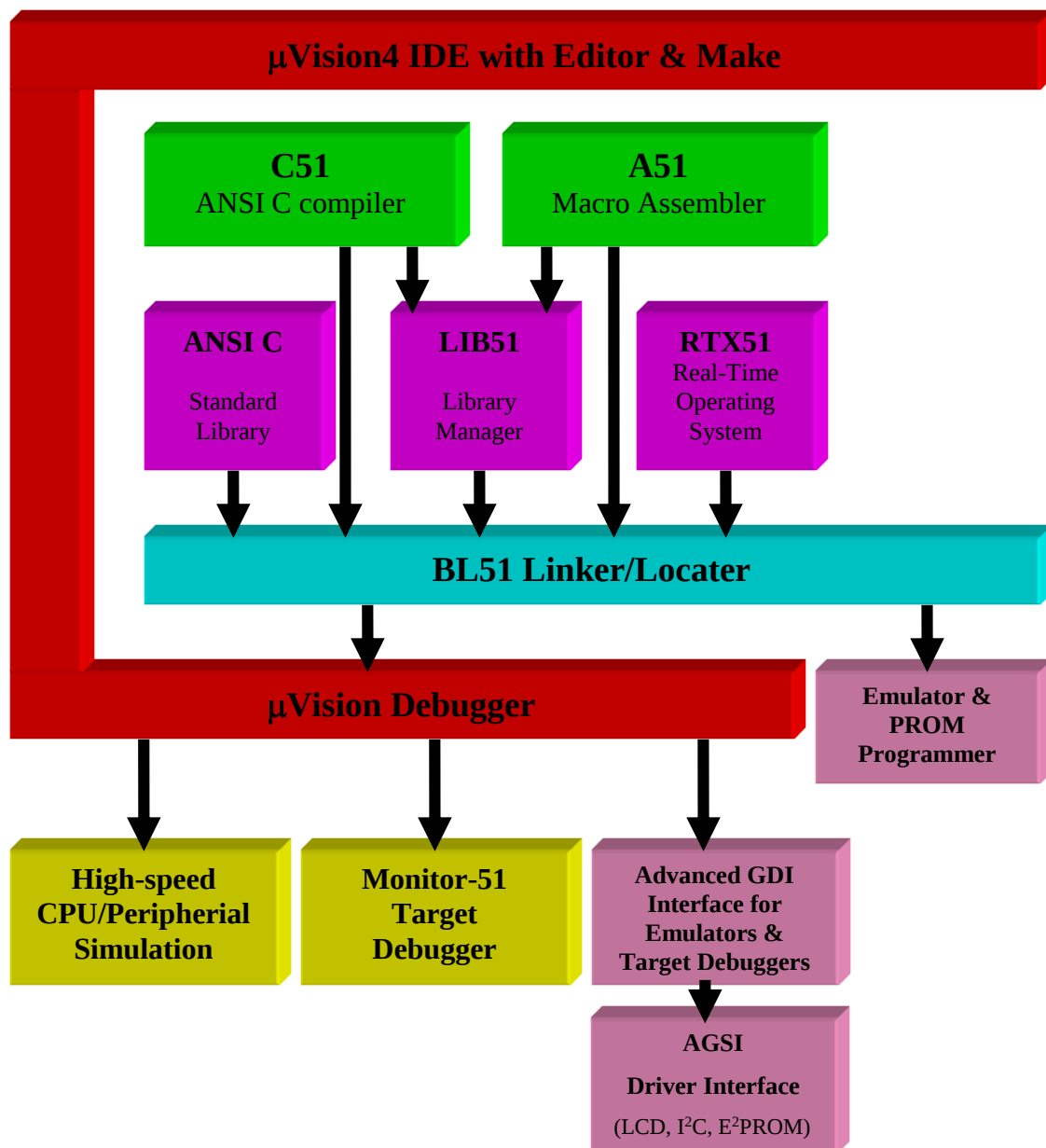
Pre správnu funkciu AGSI ovládačov je potrebné príslušné knižnice DLL nakopírovať⁴¹ do adresára C:\KEILBIN\ a doplniť v TOOLS.INI používané simulačné knižnice.

⁴¹ V niektorých prípadoch môžu mať súbory DLL odlišné názvy požadovaných súborov a je to potom potrebné správne premenovať podľa potreby μ Vision 4. Je to spôsobené staršími a neaktuálnymi AGSI na stránke, ktoré neboli dlho inovované oproti prostrediu μ Vision 4.

5 Vývojové prostriedky pre tvorbu aplikácií

5.1. Spôsoby tvorby aplikácie

Obrázok 46, Vývoj aplikácie pomocou μ Vision 4



5.2. Prehľad vývojových nástrojov

- KEIL PK-C51, PK-C251, PK-C166, PK-ARM, RL-ARM,
- Embedded Workbench,
- Embedded Workbench C++,
- MakeApp,
- Code Vision,
- Easy Code,
- Visual State,
- Version Control System,
- Rational Rose C++,
- Easy Case C, C++,
- SDL Desing Tool,
- embOS,
- RTX One⁴²,
- AS552⁴³,
- ASM251.⁴⁴

5.3. Požiadavky na vývojové prostriedky

- Vysoká univerzálnosť použitia vývojového prostriedku.
- Zápis programu v C a assembleri, prípadne pomocou napr. stavového diagramu, alebo pravdivostnej tabuľky.
- Možnosť ladenia programu na úrovni C a assembleru.
- Softwarový simulátor s možnosťou vzájomného prepojenia s prototypovou doskou cez RS232, alebo rozhranie CAN.
- Hardwarový simulátor s možnosťou použitia breakpoint-ov v ladenom programe s možnosťou sledovania obsahov registrov, portov, pamäťových miest mikroprocesora a externej pamäti, stavu sériového komunikačného rozhrania RS232, zbernice I²C, Ethernet a CAN.

⁴² Program je napísaný z dôvodu zabezpečenia vysokej efektívnosti v assembler-i a v pamäti programu zaberá cca. 400 Byte. Autorom RTX One je Ing. Eduard Jadroň.

⁴³ Program AS552 je už bez podpory výrobcu.

⁴⁴ ASM251 je jednoduchý, jednopriechodový assembler bez optimalizácie pre 16 bitový procesor rady 80251, ktorého autorom je Ing. Eduard Jadroň z Diversant Software. Program je napísaný v programovacom jazyku Turbo Pascal 7.0. Program **ASM251.EXE** obsahuje niekoľko doplnujúcich modulov, ktoré dokážu preložený program „zašifrovať“ v pamäti a výrazne tak sťažiť prípadný spätný preklad programu.

5.4. Programovací jazyk assembler A51

Na trhu existuje obrovské množstvo platených, ale aj voľne šíriteľných⁴⁵ vývojových nástrojov pre mikroprocesory 8051. Príkladom môžeme uviesť MASM51, A51, AS51⁴⁶, ASM251, AS552, 8051 Workbench, 8051 IDE a iné. Zaujímavou alternatívou je použitie špeciálnych prekladačov⁴⁷ z ASM do C, prípadne do iného jazyka. Bližšie aplikačné možnosti sú uvedené v (RTX Real Time Operating System, 2001).

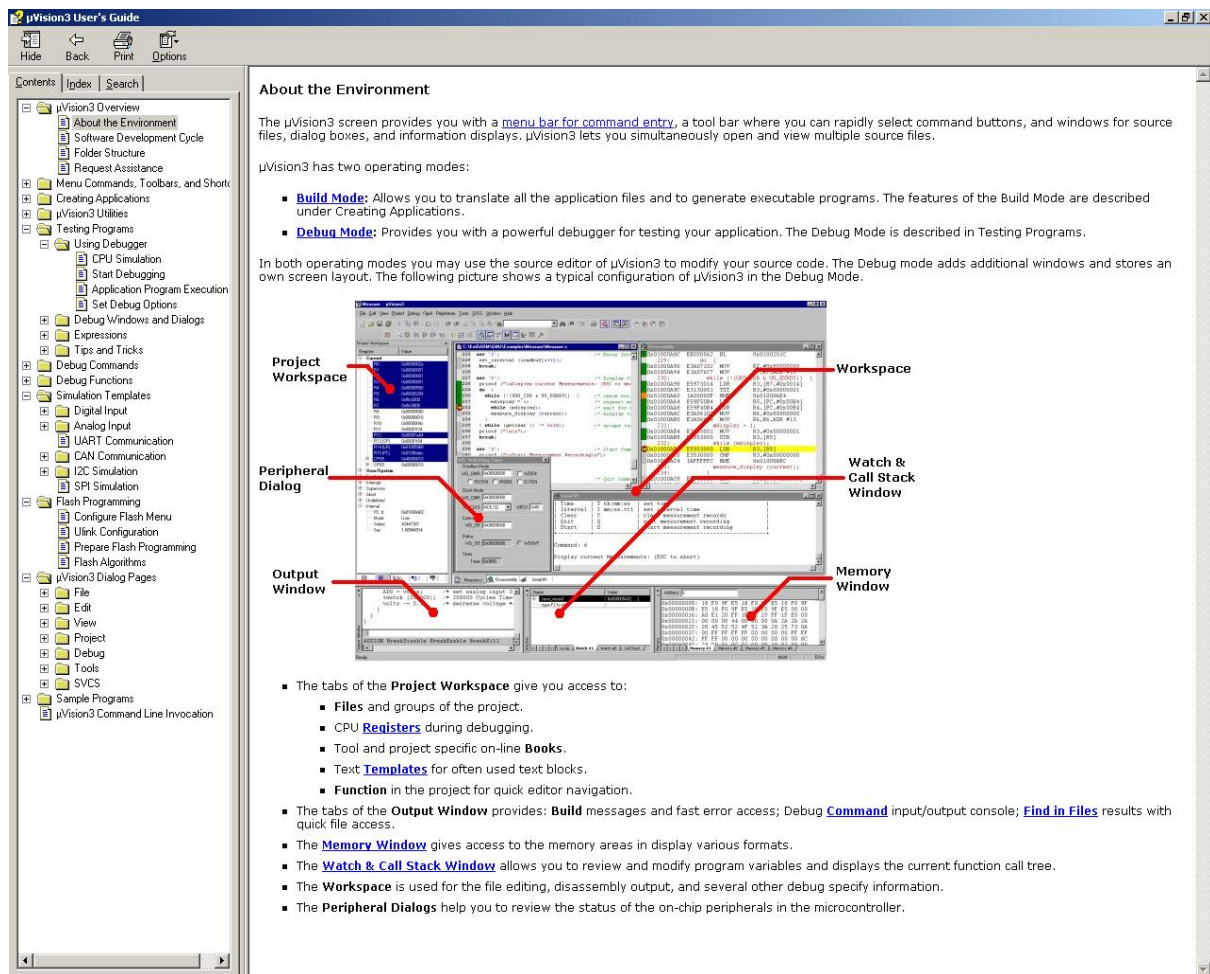
5.5. Programovací jazyk assembler ASM251 pre procesory rady 80251

⁴⁵ Shareware, alebo freeware TPtoC.

⁴⁶ Program AS51.EXE obsahuje niekoľko závažných chýb, ktoré nesprávne prekladajú niektoré presunové inštrukcie ale v novej verzii programu AS552.EXE od rovnakého výrobcu už uvedené chyby boli odstránené.

⁴⁷ <http://www.mpsinc.com/index.html>, upozorňujeme, že ide o platenú službu, kde je spoplatnený jeden riadok sumou približne 0,10 USD.

Obrázok 47, Optimálne rozdelenie pracovnej plochy vývojového nástroja

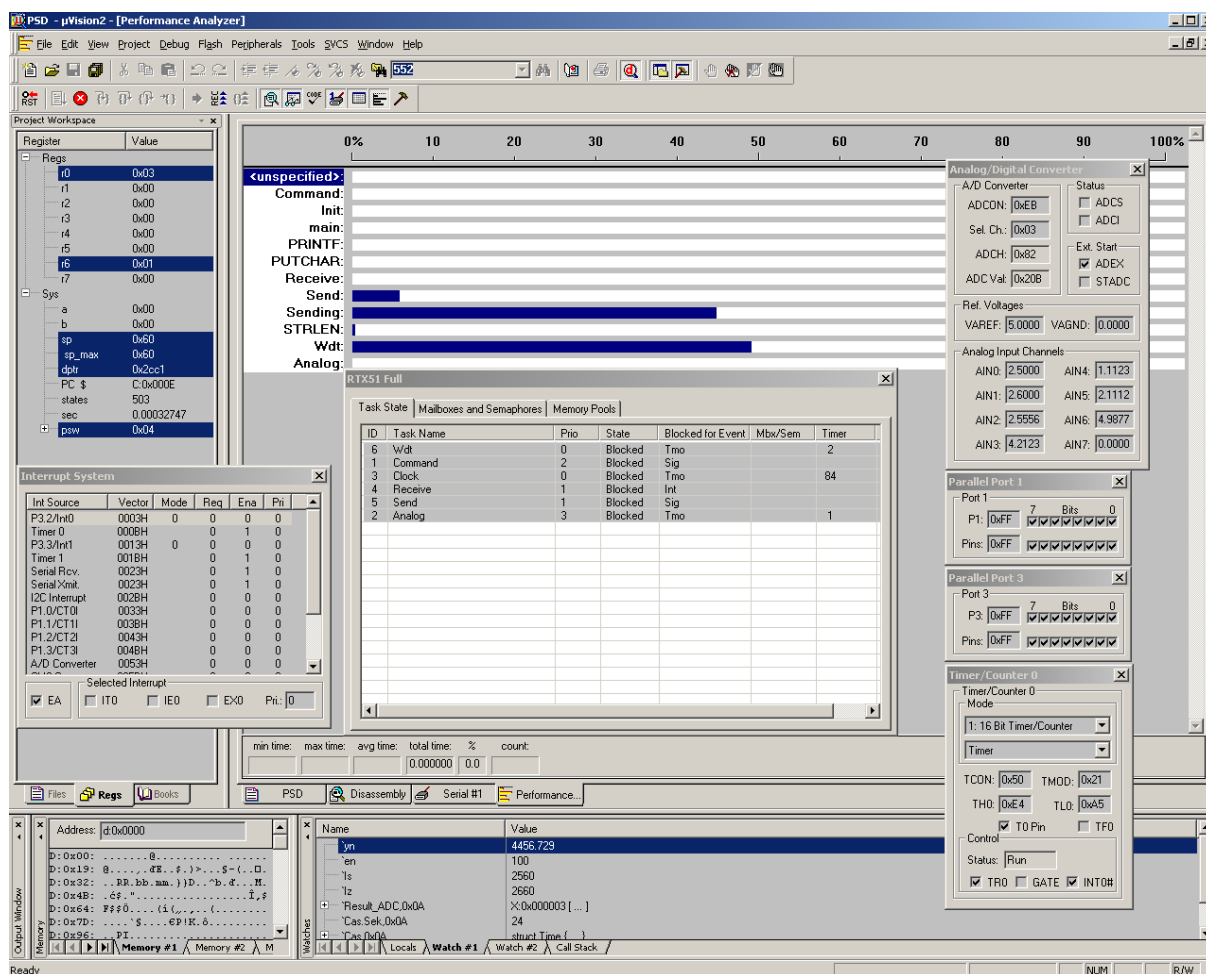


6 Prehľad vývojových nástrojov

6.1. Grafické vývojové prostredie μ Vision od firmy KEIL Software

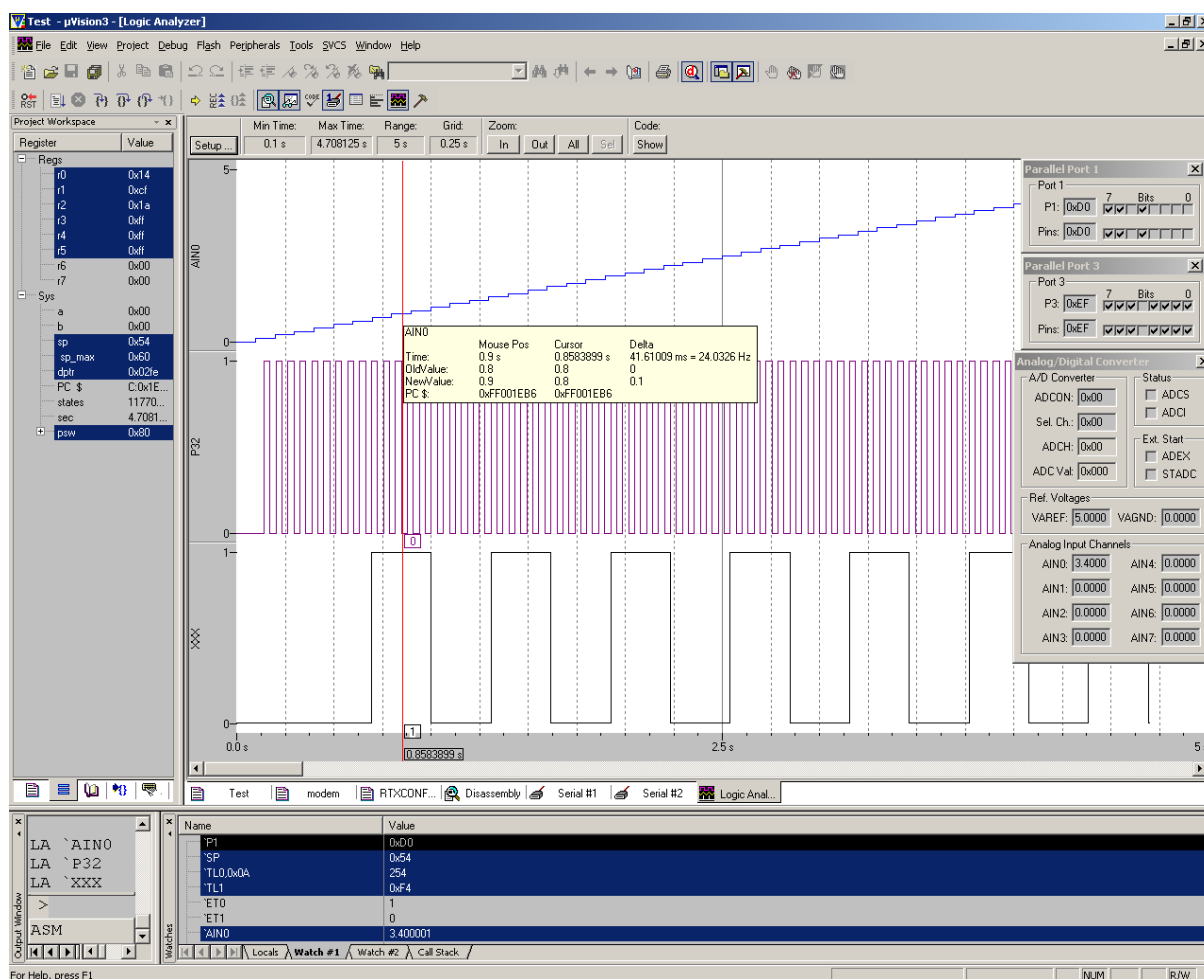
Stručná charakteristika: Prostredie μ Vision verzie 2.4 je výkonným a mocným vývojovým nástrojom pre programátora. Obsahuje integrovaný programovací jazyk C a assembler, ktorý je možné ladiť až na najnižšej úrovni assembler-u, alebo na úrovni jazyka C. Výstup sériového kanála je možné presmerovať na obrazovku, alebo na štandardný port COM1 až COM4. Programátor môže sledovať množstvo obsahov registrov, ktoré sú prehľadne blokovo usporiadané. Vývojové prostredie μ Vision je možné používať pre tvorbu aplikácií používajúce mikroprocesory 8051, 80251, 80166, ARM. Firma KEIL Software je autorom RTOS pod obchodným označením RTX51 Tiny a RTX51 Full, ktoré sa používajú na vývoj aplikácií ktoré majú pracovať v reálnom čase. Hlavnou nevýhodou tohto programu je vysoká cena. Bližšie informácie v www.keil.com.

Obrázok 48, Vývojové prostredie μ Vision 2.14 od firmy KEIL pre architektúry 80C51



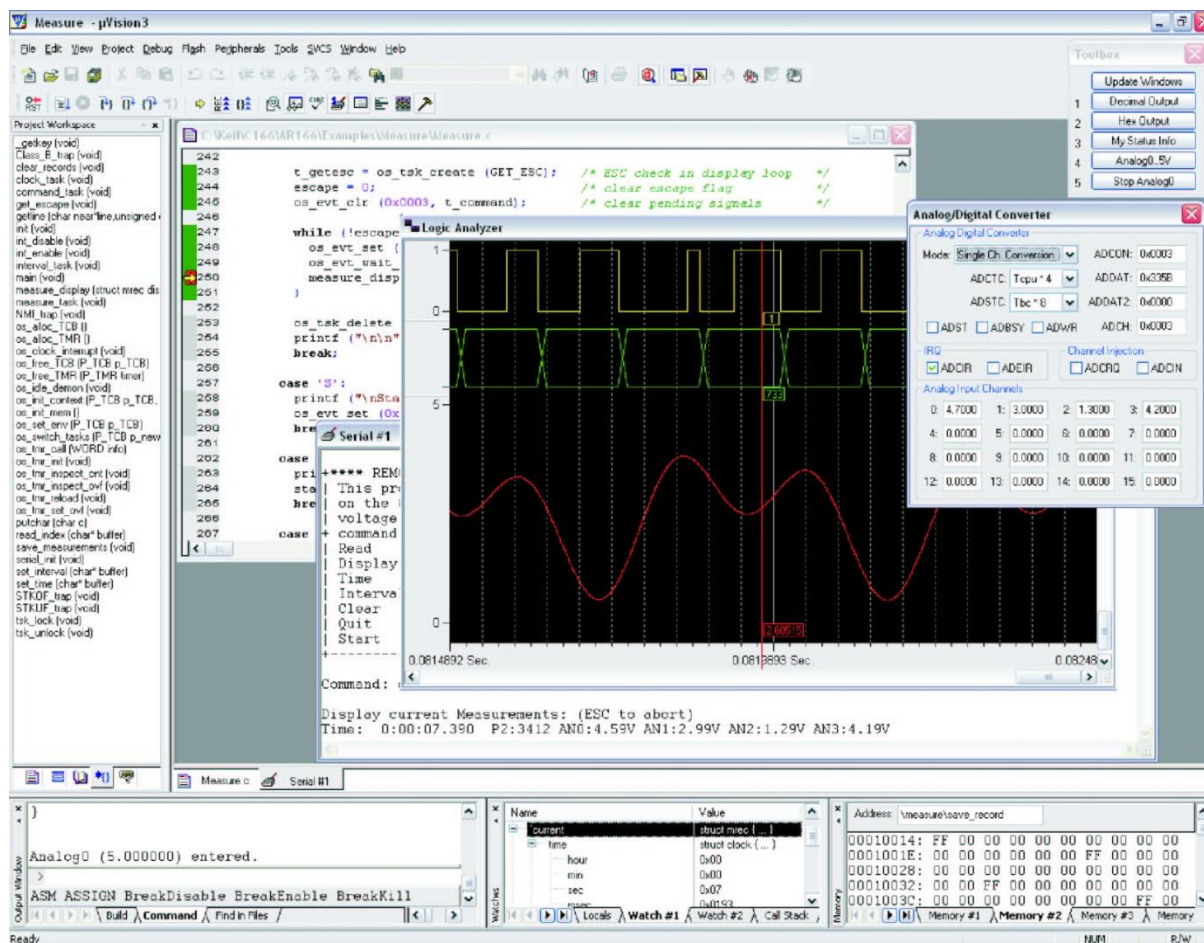
Stručná charakteristika: Vývojové prostredie μ Vision⁴⁸ 3 obsahuje okrem štandardných nástrojov aj logický analyzátor umožňujúci sledovanie analógových a číslicových signálov, čo posúva praktické využitie tohto nástroja smerom bližšie k HW emulátorom. Toto vývojové prostredie slúži na uľahčenie práce v náročných aplikáciách ako je napríklad TCP/IP Stack, rýchla Fourier-ová harmonická analýza v reálnom čase, skalárny alebo vektorový regulátor pohonu asynchrónneho motora a iných náročných aplikácií. K tomuto prekladaču je možné zakúpiť aj AR166, ktorý pomáha pri vytváraní tzv. Embedded aplikácií. Demonštračnú verziu tohto vývojového nástroja je možné získať na www.keil.com.

Obrázok 49, Vývojové prostredie μ Vision3 od firmy KEIL pre architektúry 80C51



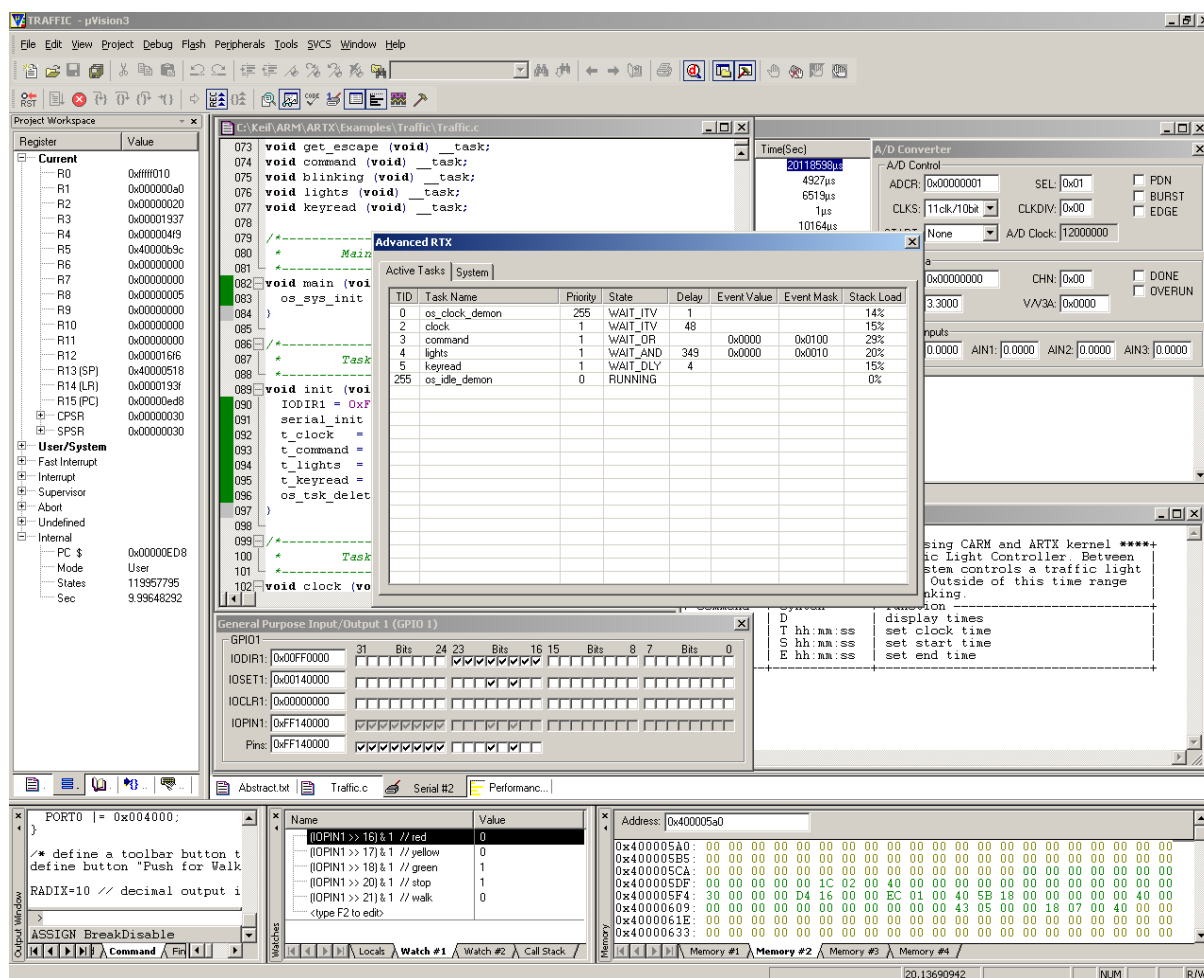
⁴⁸ Od mája 2015 je k dispozícii nová verzia PK 9.54 μ Vision 5, ktorá sa dočkala podstatných zmien ktoré zásadným spôsobom uľahčia vývoj aplikácií programátorom.

Obrázok 50, Vývojové prostredie μ Vision3 od firmy KEIL pre architektúry 80C166

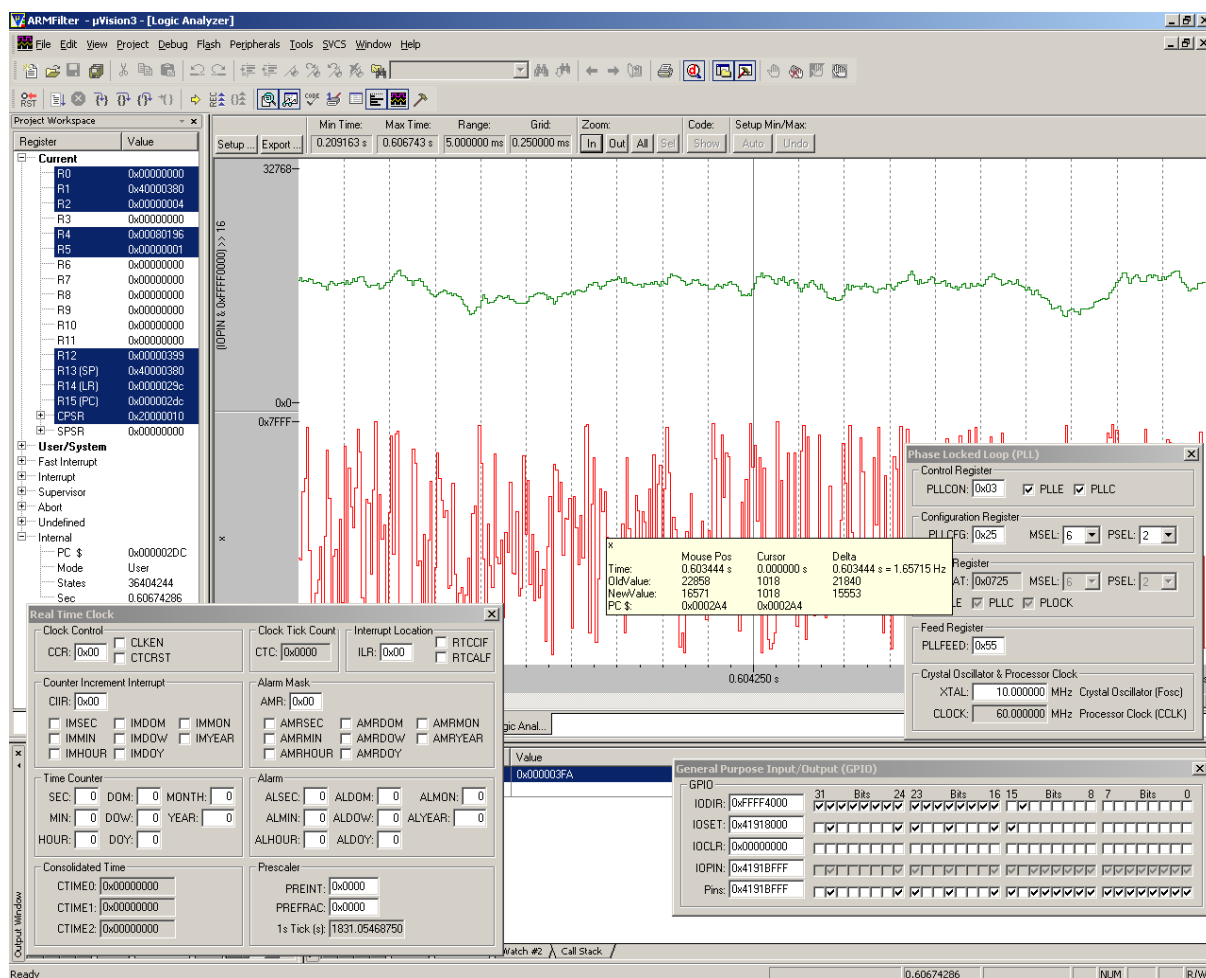


Stručná charakteristika: Toto vývojové prostredie μ Vision 3 slúži na prácu s rodinou najvýkonnejších 32 bitových mikroprocesorov ARM. Obsahuje podobné nástroje pre vývojového pracovníka a rovnaké prostredie na aké bol užívateľ zvyknutý doteraz. Výhodou tohto nástroja je fakt, že je distribuovaný ako **Public Domain** s jediným obmedzením, ktoré umožňuje ladenie programu s maximálnou veľkosťou do 16kB. Kompilátor a linker jazyka C++ je v tomto prípade bez obmedzenia a tak je možné vytvárať výsledný kód, ktorý je možné naprogramovať do mikroprocesora. K tomuto programu je dodávaný ARTX pre mikroprocesory ARM, ktorý má obdobné vlastnosti ako RTX51, alebo AR166. Demo verziu tohto vývojového nástroja je možné získať na www.keil.com.

Obrázok 51, Vývojové prostredie μ Vision3 od firmy KEIL pre architektúry ARM7TDMI

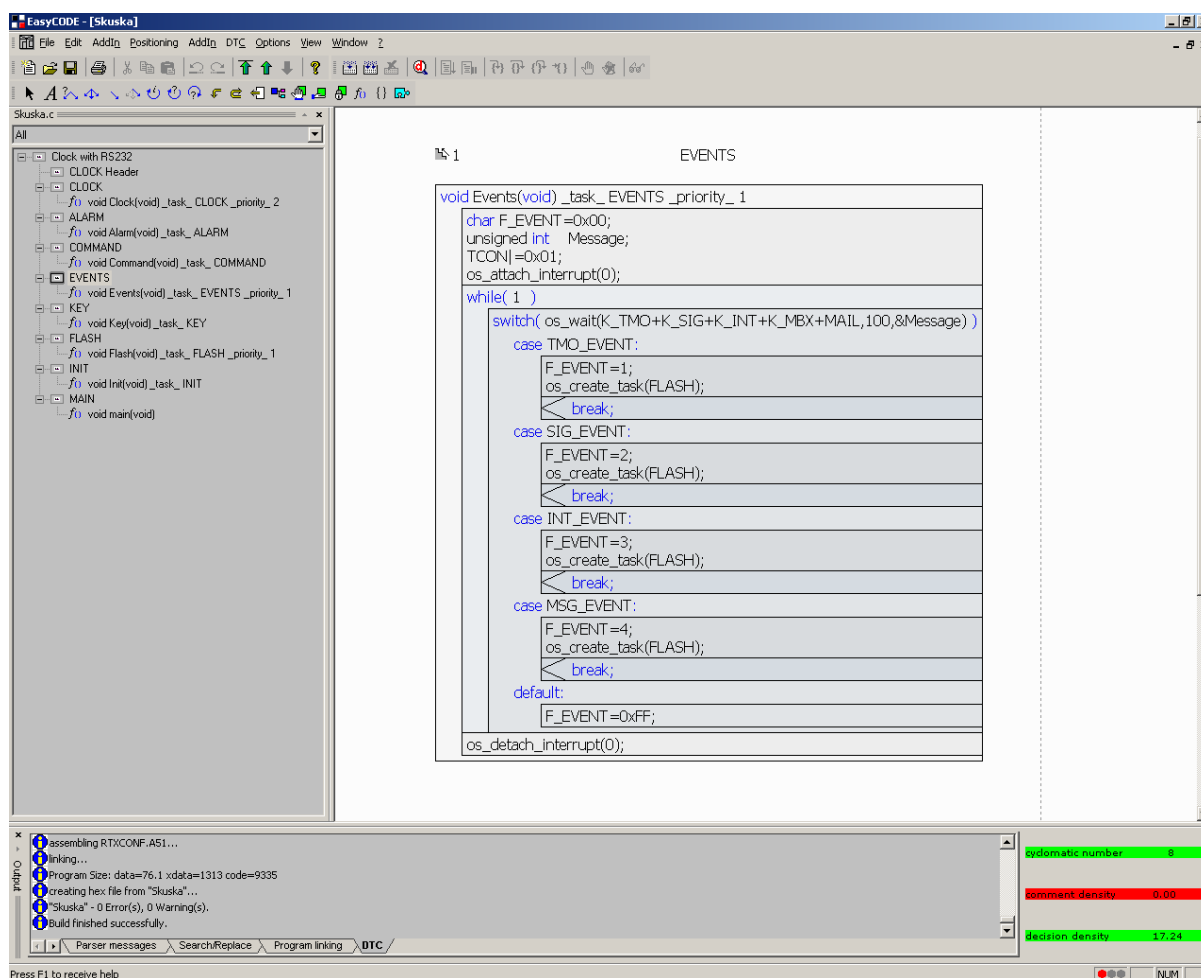


Obrázok 52, Vývojové prostredie µVision3 pre architektúry ARM



Stručná charakteristika: Vývojové prostredie EasyCODE slúži ako výkonná nadstavba pre programátora, pretože umožňuje zápis programu vo forme štruktúrovaných vývojových diagramov. Vyššie uvedený software umožňuje generovať cieľový kód pre rôzne programovacie jazyky ako je C/C++, Pascal, Cobol, Java, Basic. Takto je umožnená nezávislosť zapísaného algoritmu programu na použitom programovacom jazyku. EasyCODE využíva užívateľom používaný kompilátor a linker jazyka demo-verziu tohto vývojového nástroja je možné získať na [www⁴⁹](http://www.49) stránke a na rozdiel od ostrej verzie programu neumožňuje zápis programu na pevný disk a použitie funkcie kopírovania a iných drobných obmedzení.

Obrázok 53, Vývojové prostredie EasyCode 7.1.0.3 od firmy EasyCode Inc. pre 8051



Nespornou výhodou EasyCODE⁵⁰ je aj možnosť prepojenia s inými vývojovými prostriedkami napr. od firmy Keil, Tasking a iných pomocou unifikovaného rozhrania⁵¹ DTC,

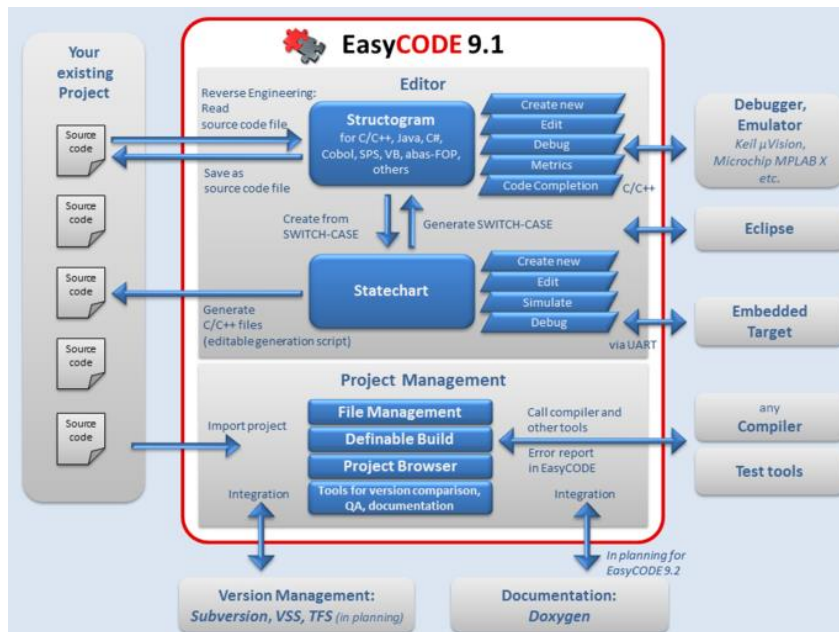
⁴⁹ www.easycode.de, www.easycode-software.com

⁵⁰ V prípade použitia je možné celý algoritmus programu vytvoriť pomocou štruktúrogramu, alebo v C jazyku napísaný program previesť do štruktúrogramu.

⁵¹ V prípade problémov s DTC je možné použiť aj rozhranie UVSOCK (The μ Vision Socket Interface) (TCP/IP) na porte 4823.

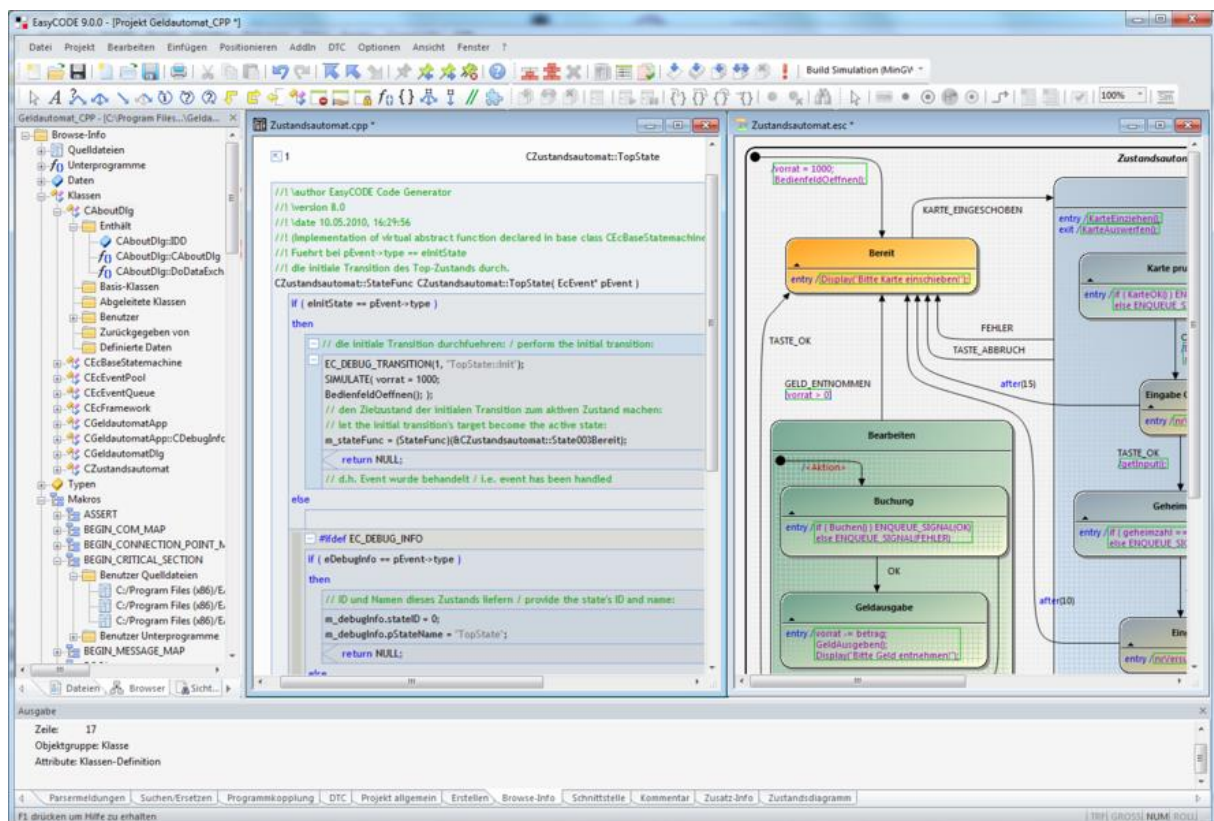
ktoré umožňuje priame ovládanie procesu kompilácie a ladenia pripojených vývojových prostredí.

Obrázok 54, Moderné možnosti programovania s EasyCODE 9.1



Zdroj: www.easycode.de

Obrázok 55, Pracovné prostredie EasyCODE 9.0.0



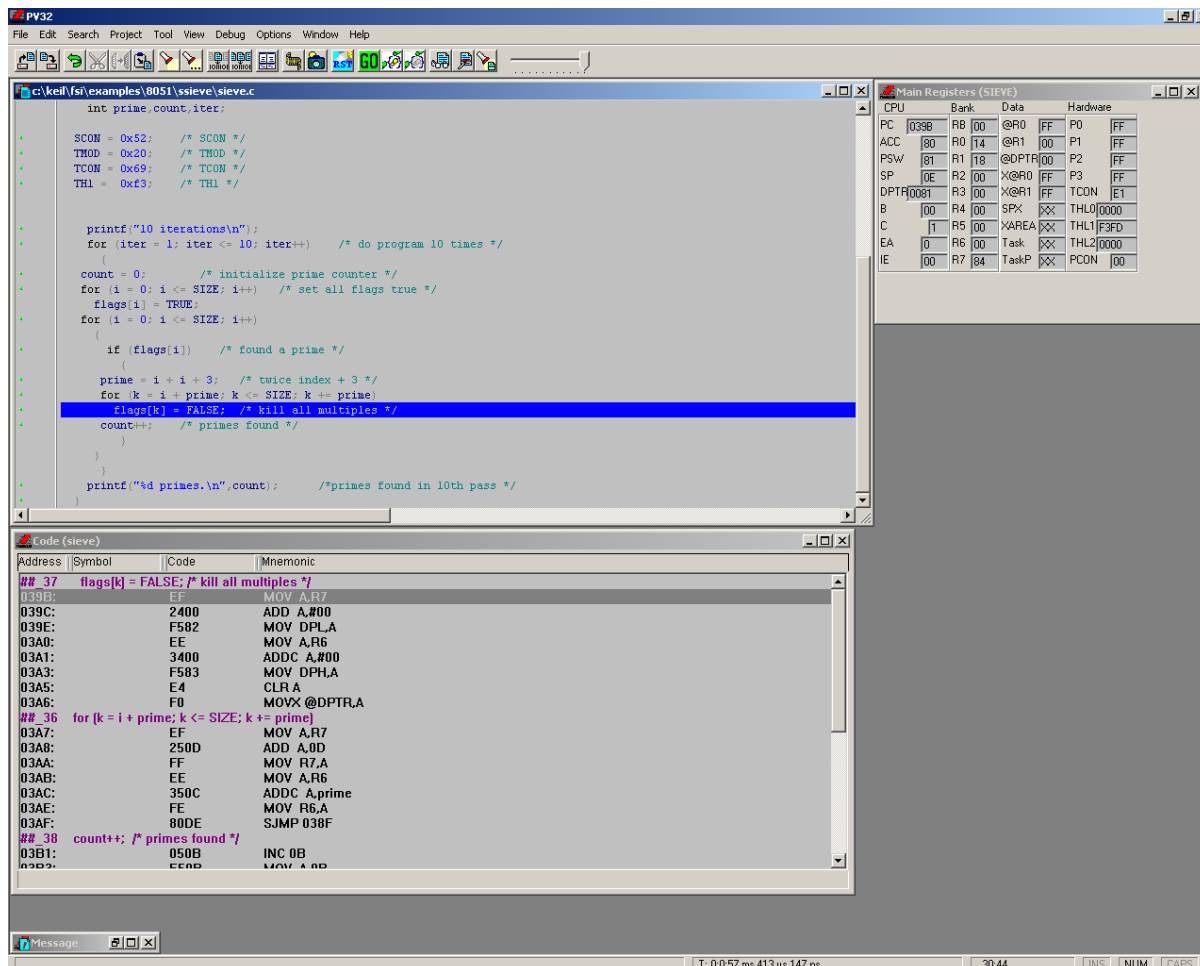
Zdroj: www.easycode.de

Prínos programu EasyCODE pre programátora je nesporný, pretože mu umožňuje vyvíjať aplikáciu pomocou vizualizačných nástrojov ktoré sú lepšie človekom prijímané. Po vizuálnom návrhu programátorom vygeneruje vysoko optimalizovaný programový kód.

6.2. Grafické vývojové prostredie ProView od Franklin Software

Stručná charakteristika: Prostredie ProView, alebo ProView32 je často používaným nástrojom profesionálnych programátorov. Umožňuje programátorovi programovať v jazyku C a assembleri. V podstate je to softwarový simulátor, ktorý nemá na rozdiel od μ Vision možnosť prepojenia programu ProView s vývojovou doskou pomocou sériového kanála. Cena tohto programu je v tomto prípade výrazne nižšia ako v prípade μ Vision. Existuje možnosť použitia RTOS v ProView od rôznych výrobcov. Na internete je možnosť získania tzv. **Evaluation version** programu, ktorú môže užívateľ po dobu 30 dní bezplatne pred zakúpením vyskúšať a potom sa rozhodnúť pre kúpu vývojového nástroja. Táto Evaluation verzia obsahuje niektoré obmedzenia, ale firma sa takýmto spôsobom účinne chráni pred nelegálnemu kopírovaniu svojich produktov.

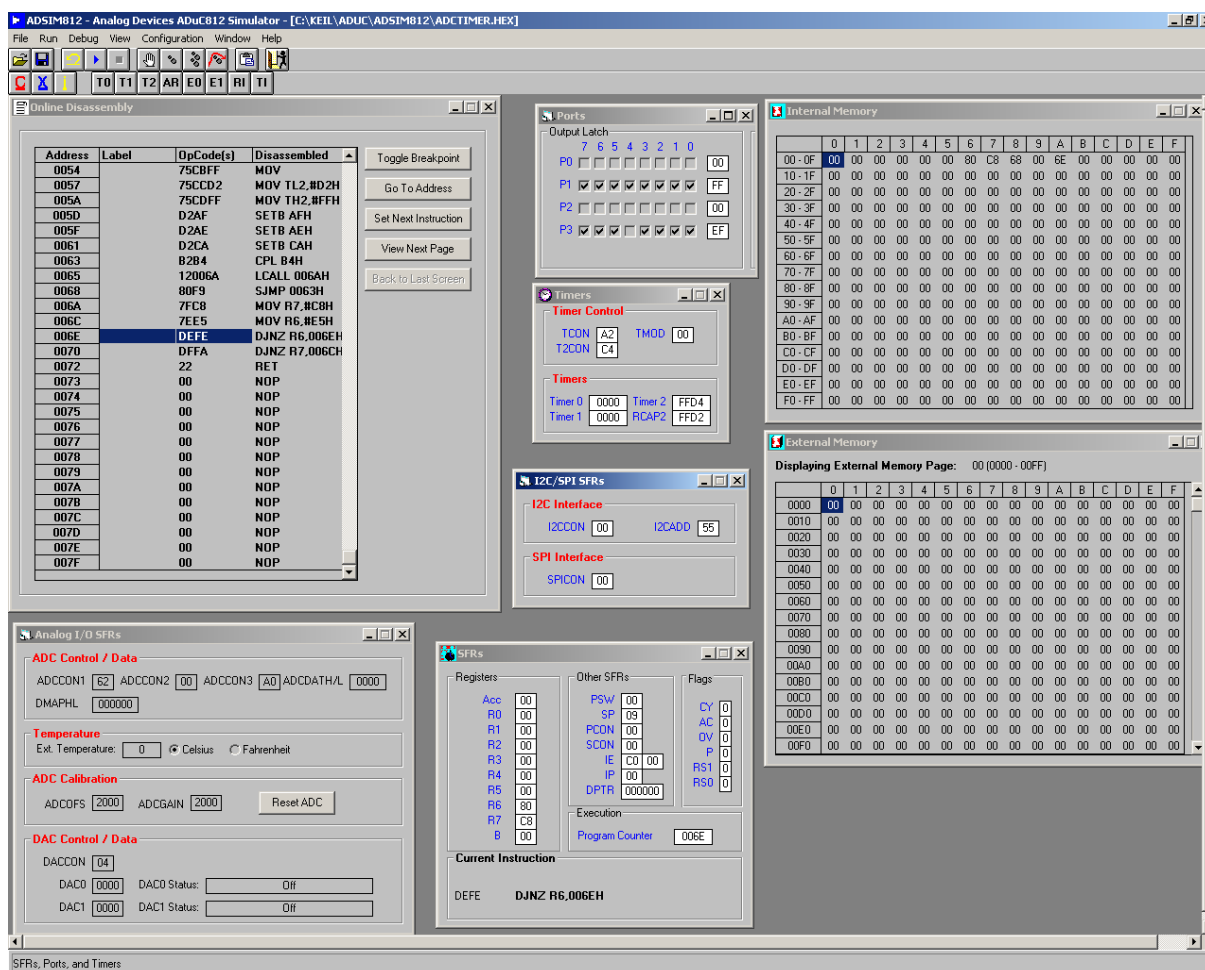
Obrázok 56, Vývojové prostredie ProView32



6.3. Grafické vývojové prostredie ADSIM812 od Analog Devices

Stručná charakteristika: Opisované vývojové prostredie umožňuje podobné funkcie ako jej konkurenčné produkty. Nespornou výhodou tohto nástroja je cena. Firma Analog Devices tento program totiž distribuuje ako freeware. Plná verzia umožňuje plnohodnotnú⁵² prácu s vývojovými mikroprocesorovými doskami.

Obrázok 57, Vývojové prostredie ADSIM812

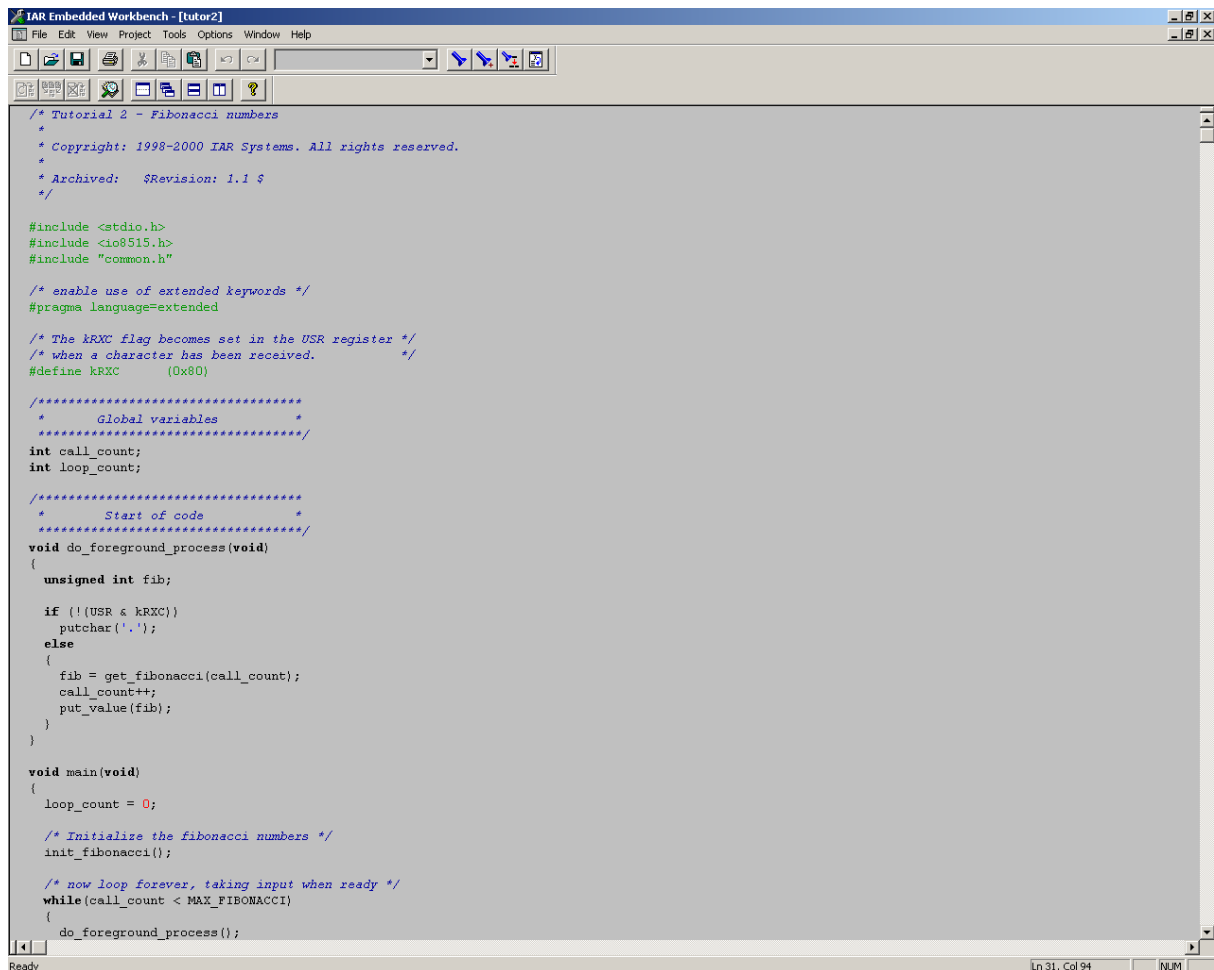


⁵² Pod pojmom „plnohodnotná“ si predstavujeme len základné a jednoduché činnosti pri programovaní aplikácií pre platformu mikroprocesorov rady 8051 bez optimalizačných a simulačných nástrojov pracujúcich na úrovni HW (hardware 8051).

6.4. Grafické vývojové prostredie Embedded Workbench od IAR Systems

Stručná charakteristika: Tento opisovaný nástroj od švédskej firmy IAR obsahuje výkonný kompilátor C a assembleru, ktorý je vybavený množstvom optimalizačných stupňov. Vybavením a funkciami je možné tento nástroj porovnať s μ Vision od firmy KEIL. Firma IAR Systems ponúka množstvo alternatívnych produktov ktoré výrazným spôsobom uľahčujú prácu programátora napr. MakeApp, VisualState, CodeVision, EasyCODE a iné. Programátor má možnosť použitia RTOS priamo vo svojich aplikáciách od rôznych výrobcov. Pre začínajúcich programátorov firma IAR uvoľnila programovací jazyk assembler bez obmedzení. Bližšie informácie na www.iar.com.

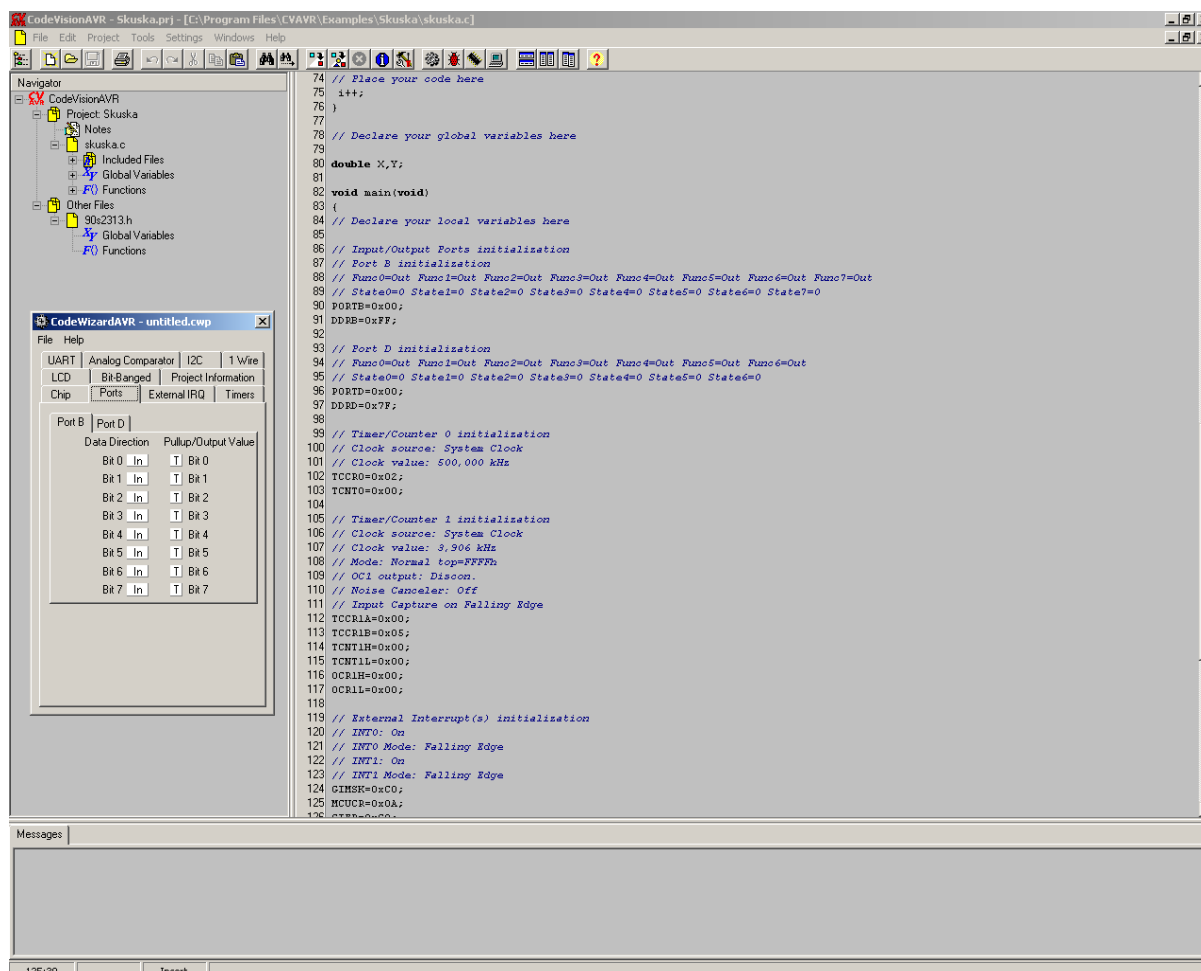
Obrázok 58, Vývojové prostredie Embedded Workbench



6.5. CodeVision AVR

Stručná charakteristika: Uvedený nástroj CodeVision sa zaraďuje do kategórie tzv. generátorov aplikácií, pretože umožňuje interaktívnym spôsobom pomocou dialógových okien nakonfigurovať jednotlivé periférne obvody mikroprocesora, čím odpadá zdĺhavé štúdium katalógových listov mikroprocesora. Výsledný kód z CodeVisionAVR môžeme priamo naprogramovať do mikroprocesora. Bližšie informácie na www.iar.com.

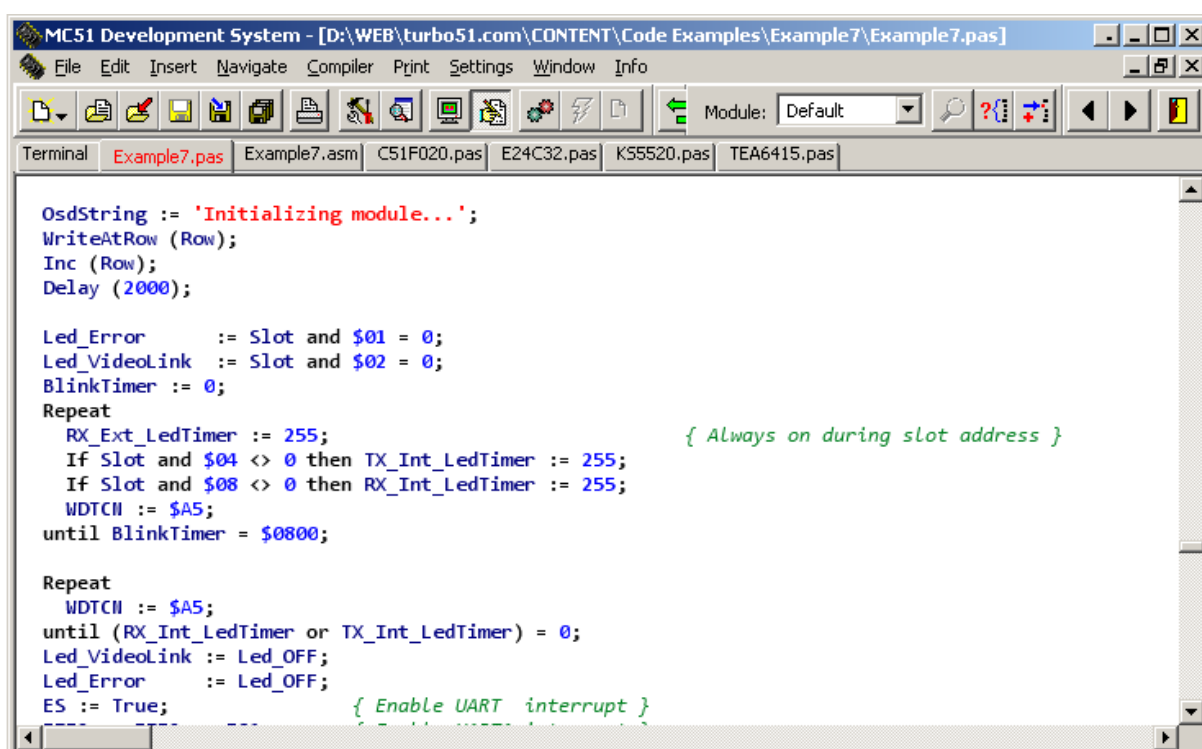
Obrázok 59, Vývojové prostredie CodeVisionAVR pre architektúry AVR



6.6. Vývojové prostredie Turbo51 pre mikroprocesory 8051

Stručná charakteristika: Tento voľne šíriteľný nástroj distribuovaný ako freeware slúži pre tvorbu aplikácií platformy 8051 v jazyku Pascal⁵³. Neobsahuje grafické prostredie, ale všetky príkazy sú zadávané pomocou príkazového riadku. Autor poskytuje aj naďalej podporu pre tento program. Kvality tohto programu sú porovnateľné s inými vývojovými prostriedkami v tejto kategórii. Aktuálna verzia Turbo51 k 30.1.2011 je 0.1.3.9. Na internete je k dispozícii veľké množstvo prekladačov jazyka Pascal⁵⁴ lišiacich sa množstvom použitých technológií, preto je na užívateľovi ktorý si zvolí pre svoju profesionálnu činnosť.

Obrázok 60, Doporučený textový IDE editor k Turbo51



⁵³ Download programu Turbo51 je možné na stránke <http://turbo51.com/download-free-pascal-compiler-8051>

⁵⁴ <http://www.thefreecountry.com/compilers/pascal.shtml>

Príklad zápisu programu v prostredí Turbo51:

```
Program Calculator ;

Const
  Osc = 22118400;
  BaudRateTimerValue = Byte (← Osc div 12 div 32 div 19200);

Var SerialPort : Text;
    Num1, Num2 : LongInt ;

Procedure WriteToSerialPort ; Assembler ;
Asm
  CLR    TI
  MOV    SBUF, A
@WaitLoop:
  JNB    TI, @WaitLoop
end ;

Function ReadFromSerialPort : Char;
Var ByteResult : Byte absolute Result;
begin
  While not RI do;
    RI := False;
    ByteResult := SBUF;

  { Echo character }

  Asm
    MOV    A, Result
    LCALL  WriteToSerialPort
  end ;
end ;

Procedure Init;
begin
  TL1 := BaudRateTimerValue;
  TH1 := BaudRateTimerValue;
  TMOD := %00100001; { Timer1: no GATE, 8 bit timer, autoreload }
  SCON := %01010000; { Serial Mode 1, Enable Reception }
  TI := True; { Indicate TX ready }
  TR1 := True; { Enable timer 1 }
end ;

{$DefaultFile On }

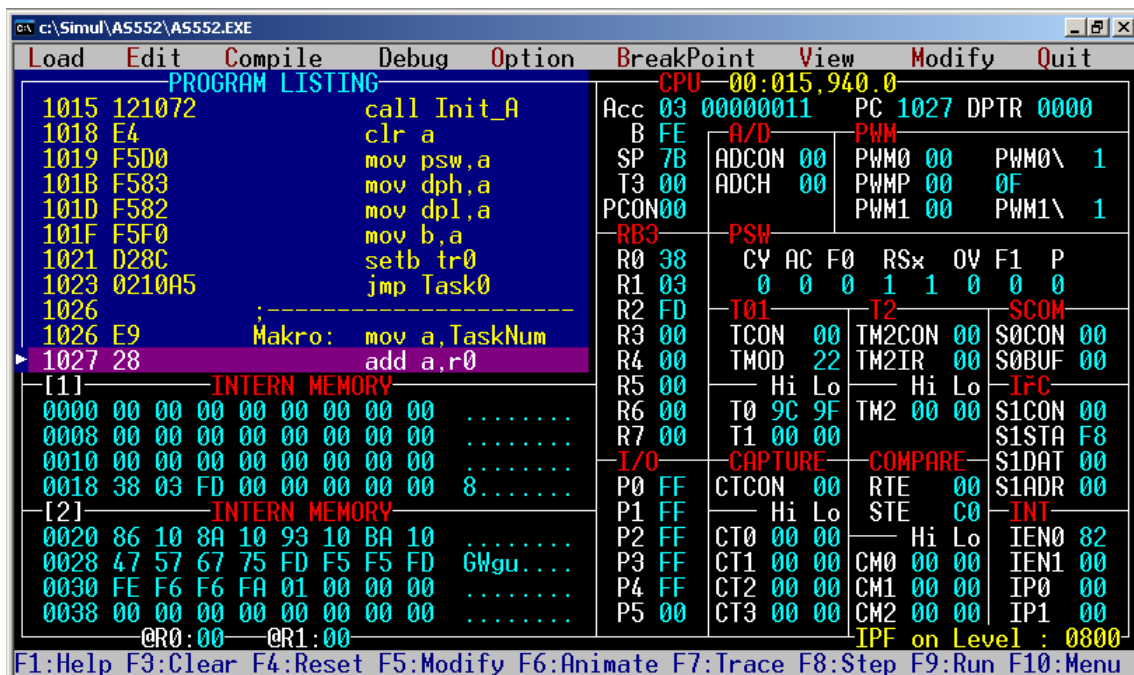
begin
  Init;
  Assign (CurrentIO, ReadFromSerialPort, WriteToSerialPort);

  Writeln ('Turbo51 IO file demo - integer calculator');
  Repeat
    Write ('Enter first number: ');
    Readln (Num1);
    Write ('Enter second number: ');
    Readln (Num2);
    Writeln (Num1, ' + ', Num2, ' = ', Num1 + Num2);
    Writeln (Num1, ' - ', Num2, ' = ', Num1 - Num2);
    Writeln (Num1, ' * ', Num2, ' = ', Num1 * Num2);
  until False;
end .
```

6.7. Vývojové prostredie AS552 od firmy Easy Soft.

Stručná charakteristika: Tento nástroj slúži na tvorbu aplikácií v jazyku assembler. Prioritne je určený pre mikroprocesory 80c552, ktoré obsahujú integrovaný 8 kanálový 10 bitový A/D prevodník. Obsahuje integrovaný prekladač⁵⁵ a simulátor⁵⁶, ktorý umožňuje sledovať činnosť vykonávaného programu a obsahu jednotlivých registrov mikroprocesora. Ovládanie programu AS552 je veľmi jednoduché a odpovedá možnostiam operačného systému MSDOS.

Obrázok 61, Vývojové prostredie AS552



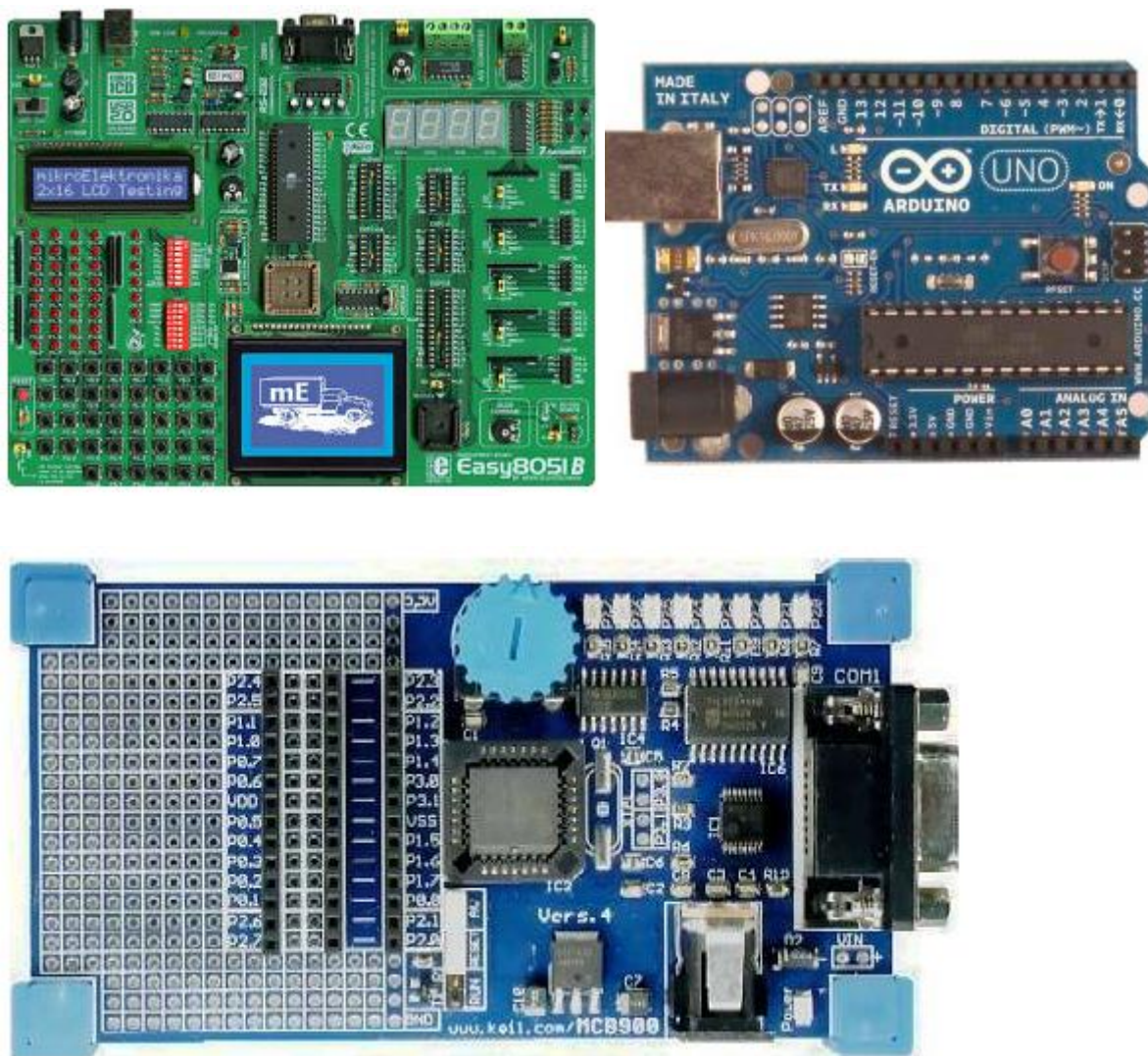
⁵⁵ Výrobca uviedol na trh aj prekladač len pre mikroprocesor 8051 pod názvom AS51, ktorý žiaľ kvôli množstvu chýb ktoré spôsobuje kompilátor v preloženom kóde nie je možné použiť pre profesionálnu aplikačnú oblasť.

⁵⁶ View51 umožňuje simulovať prítomnosť logických úrovní na jednotlivých bitoch a portoch procesora 80c552 a zobraziť ich na časovej osi.

7 Hardware pre vývojové aplikácie

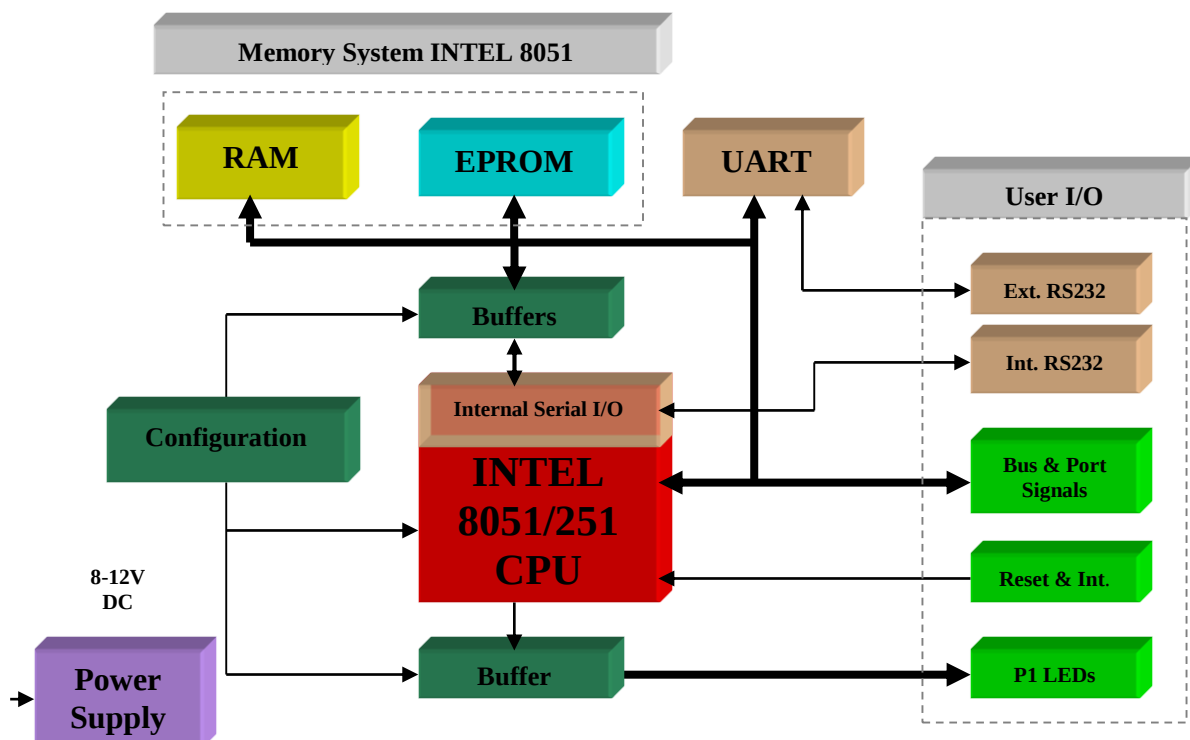
Rôznorodosť aplikácií používaných v spojitosti s mikroprocesorovou technikou má za následok množstvo rôznych vývojových dosiek na trhu, ktoré existujú v rôznych konfiguráciách. Každá aplikácia si vyžaduje špecifický hardware, ktorý umožňuje plniť svoju primárnu činnosť. Táto kapitola jednoduchým spôsobom čitateľa oboznámi s jednoduchými vývojovými doskami ktoré sú dostupné na trhu a sú schopné spolupracovať s vývojovým prostredím.

Obrázok 62, Hardware pre mikroprocesory 8051



7.1. Hardware 80C51

Obrázok 63, Aplikačný model mikroprocesorového systému s 80Cx51

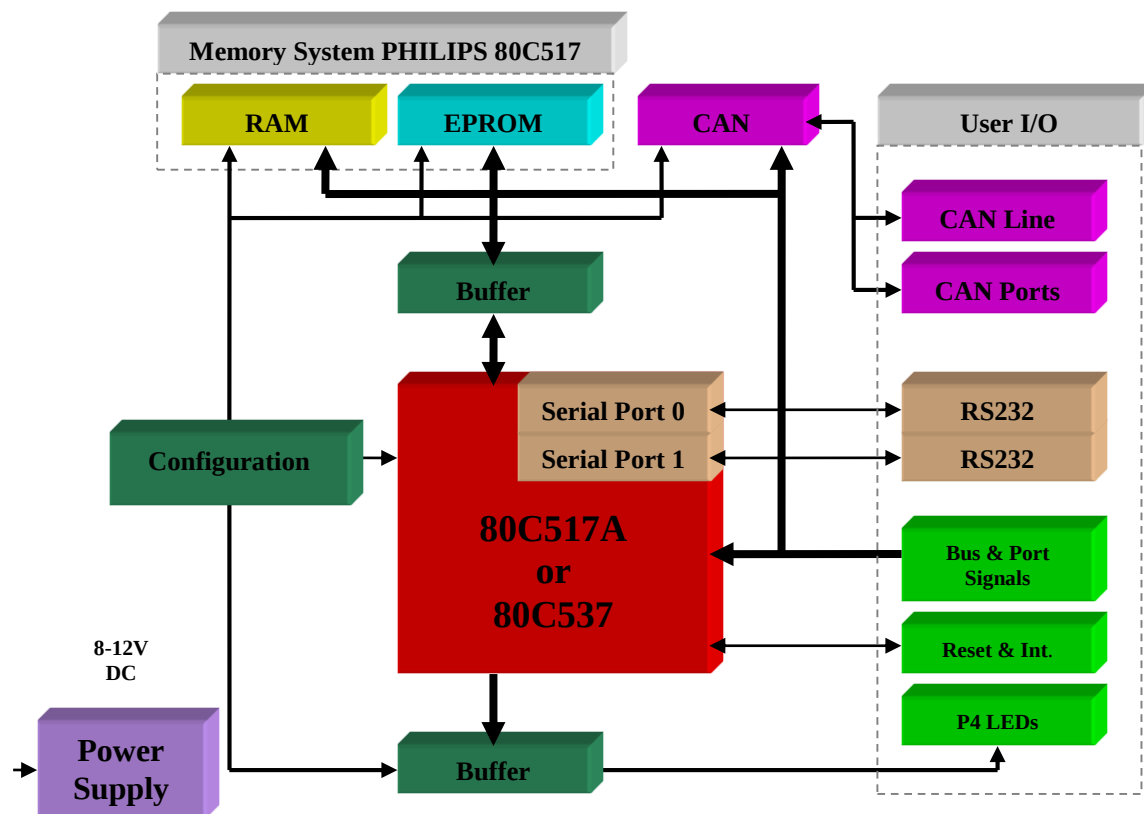


Obrázok 64, Vývojová doska s mikroprocesorom Intel 80Cx51



7.2. Hardware 80C517

Obrázok 65, Aplikačný model mikroprocesorového systému s 80C517

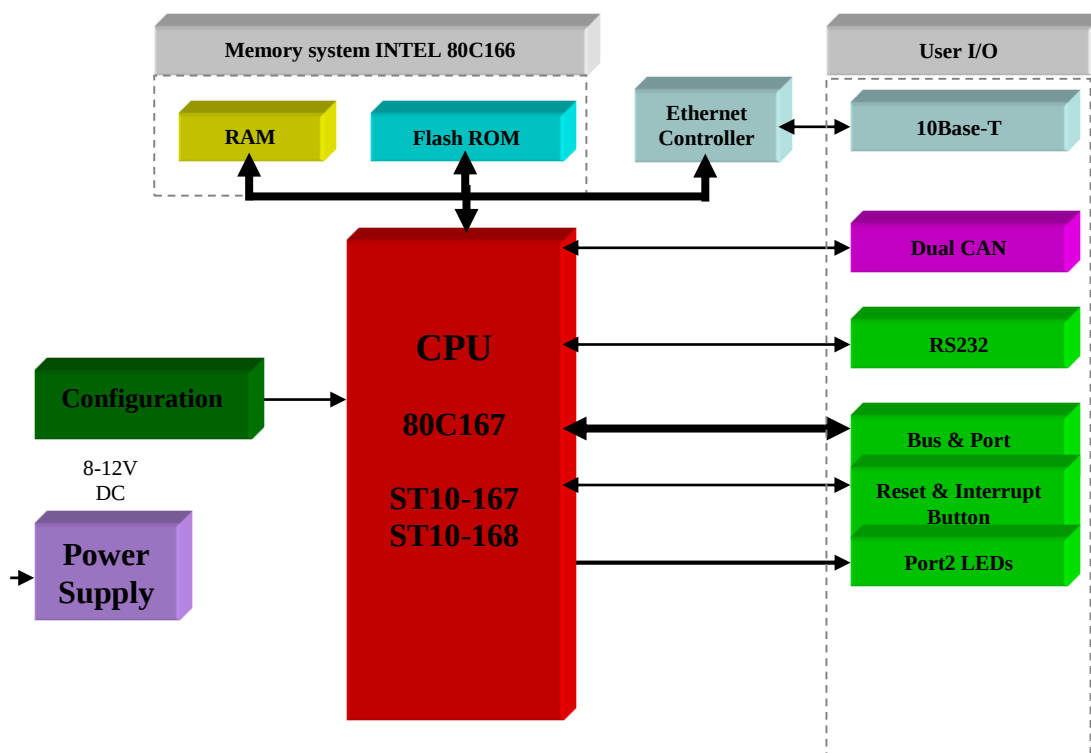


Obrázok 66, Vývojová doska s mikroprocesorom Siemens 80C517

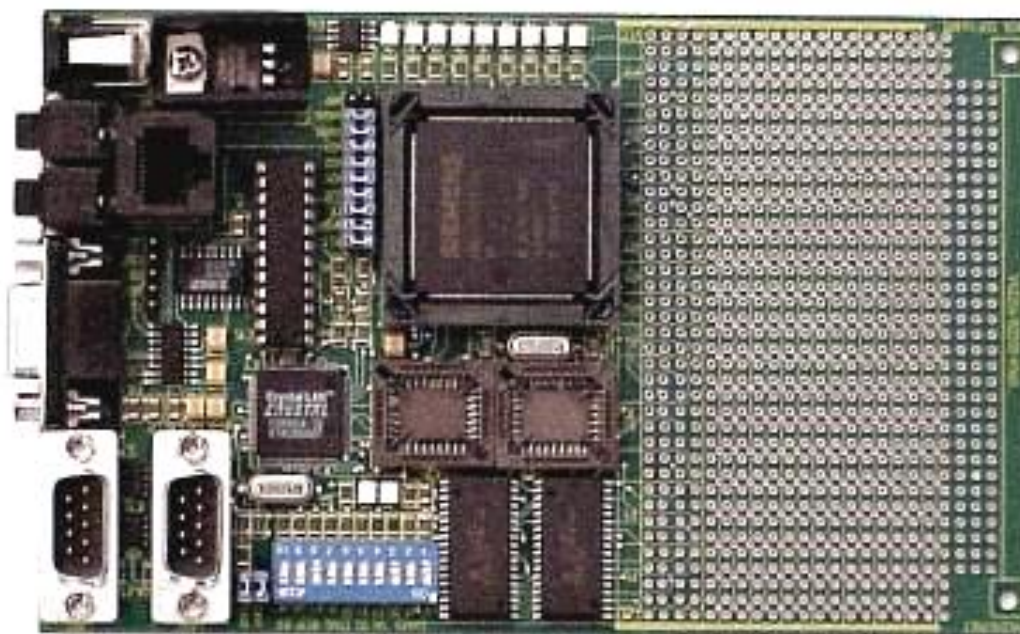


7.3. Hardware 80C166

Obrázok 67, Aplikačný model mikroprocesorového systému s 80C166

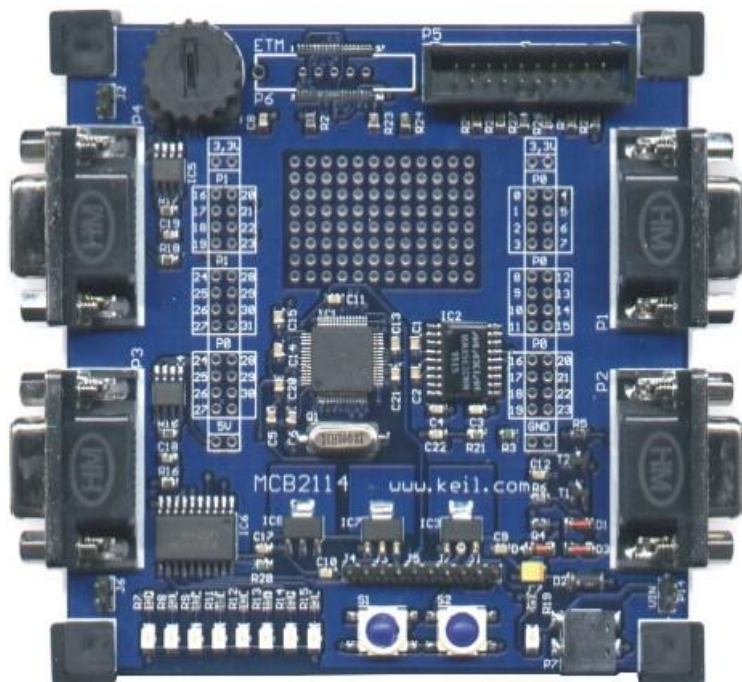


Obrázok 68, Vývojová doska s mikroprocesorom Siemens 80C166



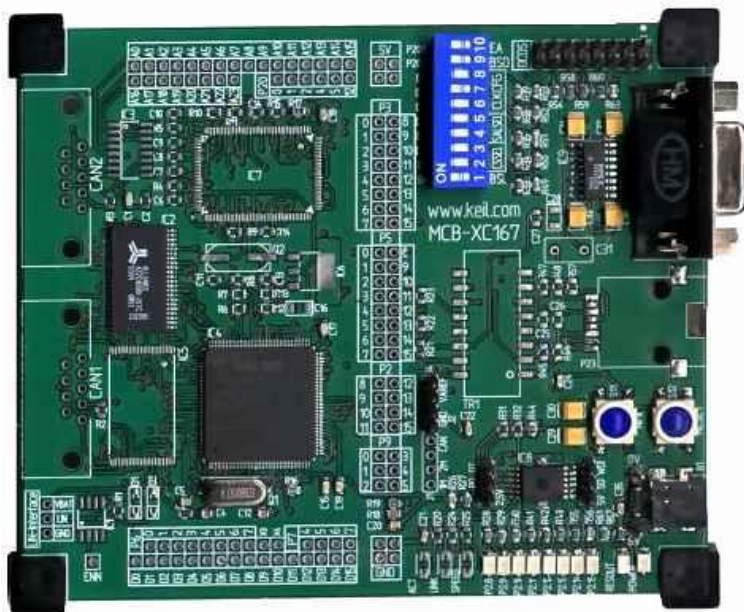
7.4. Hardware MCB2100

Obrázok 69, Vývojová doska s mikroprocesorom Philips LPC2129



7.5. Hardware MCBXC167

Obrázok 70, Vývojová doska s mikroprocesorom Infineon XC167CI



8 Systémy pracujúce v reálnom čase RTOS

Niektoré mikroprocesory majú pracovať v prostredí kde sa vyžaduje súčasné vykonávanie viacerých úloh. Dobrým príkladom je pravidelný zber dát z rôznych snímačov. Dáta môžu prichádzať do mikroprocesora v poradí ktoré môžeme nazvať chaotické. Snímače môžu poskytovať hodnotu nameranej veličiny mikroprocesoru ako binárne číslo, napätie, frekvencia, alebo šírka impulzu. Mikroprocesor v aplikácii má spracovať prijaté dáta zo snímačov, vykonať zásah v regulovanej sústave a zobrazit' stav sústavy na zobrazovacej jednotke. Z vyššie uvedeného vidíme, že mikroprocesor musí vykonať viacero úloh za jednotku času.

Doteraz musel programátor venovať množstvo pozornosti cyklickej⁵⁷ kontrole výskytu očakávaných udalostí v uzavretej programovej slučke čo kládlo zvýšené nároky na schopnosti programátora vytvoriť efektívny kód.

S pomocou operačného systému RTX51 je možné veľmi jednoducho vytvoriť program ktorý sa bude skladať z požadovaného počtu elementárnych úloh pričom každej úlohe bude vopred predpísaná udalosť ktorá ju zaktivuje. Úlohy sú zároveň schopné vzájomnej výmeny dát, čo zaručuje ich kompaktnosť a konzistenciu, takže programátor môže venovať maximálnu pozornosť tvorbe obslužného programového kódu programu.

Reálne operačné systémy RTOS (*Real Time Operating System*) sú špeciálne operačné systémy vytvorené tak aby umožňovali rýchlu reakčnú dobu, spoľahlivosť a úsporné zaobchádzanie so systémovými zdrojmi CPU.

RTOS rapídny spôsobom urýchľuje vývoj aplikácií pretože obsahuje funkcie ktoré umožňujú mnoho vecí vykonávať jednoduchšie, rýchlejšie a efektívnejšie.

⁵⁷ Niekedy je tento spôsob obsluhy zariadení nazývaný ako „pooling“.

Tabuľka 3, Charakteristické vlastnosti RTOS od firmy KEIL

Description Part Number	RTX51 FULL FR51	RTX51 TINY Included with PKS1
Parametric Specifications		
Round-Robin Multitasking	✓	✓
Preemptive Multitasking	✓	
Cooperative Multitasking	✓	✓
Timeout Events	✓	✓
Interval Events	✓	✓
Signal Events	✓	✓
Message Events	✓	
Semaphore Events	✓	
Memory Pools	✓	
Code Banking Support	✓	
<u>CAN Libraries</u> (for the Intel 82526 and 82527; Philips 82C200 and 8xC592; and Infineon 81C90, 81C91, C505C, and C515C)	✓	
Maximum Number of Defined Tasks	256	16
Maximum Number of Active Tasks	19	16
Required CODE Space	6K-8K bytes	900 bytes
Required DATA Space	40-46 bytes	7 bytes
Required Stack (IDATA) Space	20-200 bytes	3 bytes for each task
Required XDATA Space	650 bytes (minimum)	0 bytes
Timer Used	0, 1, or 2	0
System Clock Divisor	1,000-40,000 cycles	1,000-65,535 cycles
Interrupt Latency	Less than 50 cycles	Less than 20 cycles
Context Switch Time - Fast Task (Depends on Stack Load)	70-100 cycles	
Context Switch Time - Standard Task (Depends on Stack Load)	180-700 cycles	100-700 cycles
Task Priority Levels	4	1
Maximum Number of Semaphores (Binary)	8	
Maximum Number of Mailboxes	8	
Mailbox Size	8 entries	
Maximum Number of Memory Pools	16	

Tabuľka 4, Časovanie a vlastnosti RTOS AR166 Kernel V1.00 od firmy KEIL

Timing Specifications for AR166 Kernel V1.00

Function	Small	Compact	Medium	Large	HCompact	HLarge	XLarge
Initialize system (os_sys_init), start task	23.6	28.3	23.9	28.5	28.3	28.5	30.1
Create defined task, no task switch	8.0	10.3	8.0	10.3	10.3	10.3	12.2
Create defined task, switch task	11.6	14.3	11.8	14.5	14.3	14.5	16.0
Delete task (os_tsk_delete)	3.0	5.8	3.0	5.8	5.8	5.8	6.8
Task switch (by os_tsk_delete_self)	4.6	6.5	4.8	6.7	6.6	6.8	6.5
Task switch (by os_tsk_pass)	3.9	5.2	4.0	5.3	5.2	5.3	5.8
Task switch (upon set event)	4.7	6.3	5.1	6.7	6.3	6.7	7.1
Task switch (upon sent semaphore)	4.3	6.3	4.7	6.7	6.3	6.8	7.6
Task switch (upon sent message)	4.7	7.1	5.1	7.5	7.2	7.6	8.7
Set event (no task switch)	0.9	1.3	0.9	1.3	1.3	1.3	1.2
Send semaphore (no task switch)	0.6	1.0	0.6	1.0	1.0	1.0	1.0
Send message (no task switch)	1.1	1.8	1.1	1.8	1.8	1.8	2.5
Get own task identifier (os_tsk_self)	0.4	0.6	0.4	0.6	0.6	0.6	0.4
Interrupt response for ISR (with using-atr.)	0.3	0.3	0.3	0.3	0.3	0.3	0.2
Interrupt lockout for ISR's	0.1	0.1	0.1	0.1	0.1	0.1	0.1
Interrupt latency for ISR's (response + lockout)	0.4	0.4	0.4	0.4	0.4	0.4	0.3

1. All times are in [μs], valid for XC16x CPU with the system clock 40 MHz, the code is executed from internal flash
2. The AR166 Kernel is configured for: 10 tasks, 10 user timers
3. These timings are measured with uVision3 debugger, simulation S166.DLL ver.: 2.42
4. Interrupt response for ISR's includes the ISR prolog as generated by C166. This prolog does the register bank switching (C166 interrupt procedure with "_using_" attribute). Note that C166 interrupt procedures have the same interrupt response time without AR166 Kernel.
5. Task switching includes swapping of the currently used stack contents. These measurements are done with 6-8 stack bytes.

Tabuľka 5, Charakteristické vlastnosti AR166 a RTX166Tiny

AR166 Kernel Technical Data

Description	AR166 Kernel	RTX166 Tiny
Number of tasks	256 max.	32 max.
Number of mailboxes	not limited	Not supported
Number of semaphores	not limited	Not supported
Number of signals	16 event flags per task	32 max.
Number of user timers	not limited	Not supported
RAM Requirements	Min. 500 Bytes	8 + 4 * number of tasks Bytes
Code requirements	Less than 4 KBytes	Less than 1.5 KBytes
Hardware Requirements	One on-chip timer (any timer may be used)	One on-chip timer (any timer may be used)
User task priorities	1 - 127	Not supported
Context switch time	Less than 25μsec @20MHz, 0 ws.	40 - 100 μsec
Interrupt lockout time	0.2 μsec	Less than 4 μsec, 0 waitstates

1. Items, which are flagged **not limited**, are not limited by the AR166 Kernel OS, however they are limited by the available system memory resources.
2. Signals are named **events** for AR166 Kernel. Each task has available 16 event flags to wait for.
3. Ram requirements depend on the number of concurrent running tasks.

Tabuľka 6, Charakteristické vlastnosti ARTX pre mikroprocesory ARM

ARTX Kernel Technical Data

Description	ARTX Kernel
Number of tasks	256 max.
Number of mailboxes	not limited
Number of semaphores	not limited
Number of signals	16 event flags per task
Number of user timers	not limited
RAM Requirements	Min. 500 Bytes
Code requirements	Less than 5 KBytes
Hardware Requirements	One on-chip timer (any timer may be used)
User task priorities	1 - 255
Context switch time	Less than 5μsec @60MHz, 0 ws.
Interrupt lockout time	1.8 μsec @60MHz, 0 ws.

1. Items, which are flagged **not limited**, are not limited by the ARTX Kernel OS, however they are limited by the available system memory resources.
2. Signals are named **events** for ARTX Kernel. Each task has available 16 event flags to wait for.
3. Ram requirements depend on the number of concurrent running tasks.

Tabuľka 7, Prehľad vývojových nástrojov od firmy KEIL

Development Tools	Part Number			
	<u>PK51</u>	<u>DK51</u>	<u>CA51</u>	<u>A51</u>
far Code/Data Support Dallas 390, Philips MX, Analog Devices ADuC812	✓			
µVision2 IDE	✓	✓	✓	✓
AX51 Ext. Macro Assembler	✓			
CX51 Ext. C Compiler	✓			
LX51 Ext. Linker	✓			
OHX51 Ext. OBJ-HEX Converter	✓			
A51 Macro Assembler	✓	✓	✓	✓
BL51 Code Banking Linker	✓	✓	✓	✓
OH51 OBJ-HEX Converter	✓	✓	✓	✓
OC51 Banked OBJ Converter	✓	✓	✓	
C51 ANSI C Compiler	✓	✓	✓	
µVision2 Debugger	✓	✓		
MON51 Target Monitor	✓	✓		
MON390 Target Monitor	✓			
ISD51 In-System Debugger	✓			
RTX51 Tiny Real-time Kernel	✓			

8.1. Opis funkcie RTOS

Aby sme pochopili význam RTOS⁵⁸, musíme si predstaviť aplikáciu ktorá má vykonávať súčasne niekoľko čiastkových úloh:

- ✓ regulovať pohon,
- ✓ snímať skutočnú hodnotu prúdu pohonu,
- ✓ snímať a vyhodnocovať teplotu pohonu,
- ✓ vykonávať funkciu regulátora pre pohon,
- ✓ obsluhovať klávesnicu a LCD display,
- ✓ komunikovať s okolím pomocou rozhrania RS232.

Z vyššie uvedeného je viditeľné, že je možné oddeliť funkciu regulátora pohonu do niekoľkých čiastkových úloh, alebo samostatných modulov. Efektívnejším spôsobom by bolo napísať program pre každú úlohu a tak vytvoriť požadované funkcie. Každý modul potom obhospodaruje samostatné premenné a funkcie, ktoré sú zapuzdrené a tak neviditeľné z okolia. Problém nastáva vtedy, ak potrebujeme vzájomne odovzdávať namerané dáta medzi jednotlivými modulmi ktoré sú spustené na rovnakej CPU. Práve v týchto činnostiach nám pomáha RTOS, kde je možné vopred definovať počet požadovaných úloh – taskov.

RTOS potom spravuje všetky úlohy a ich systémové zdroje ako je pamäť, priority a dáta. Všetky úlohy spustené v RTOS dostanú od plánovača (*scheduler*) časový úsek v priebehu ktorého sú aktívne. Plánovač môže za niektorých okolností prerušiť vykonávanie úlohy ak je potrebné vykonať obsluhu prerušenia, alebo vykonať úlohu s vyššou prioritou, ale po vykonaní musí vrátiť CPU prerušenému procesu. S dopĺňujúcimi prostriedkami je možné vzájomne medzi úlohami komunikovať, synchronizovať a odovzdávať údaje.

⁵⁸ http://www.youtube.com/watch?feature=player_embedded&v=CGkl97Yghmo

8.2. Výhody použitia RTOS

- Encapsulácia - zapuzdrenie funkcií a dát v jednotlivých úlohách,
- Modulárna stavba, možnosť testovania a doplnenia systému o nové funkcie,
- Lepšia prehľadnosť a zrozumiteľnosť programu,
- Použitie funkcií v prostredí RTOS zabezpečí definované reakčné doby úloh,
- Spracovanie úloh je vykonávané vo vnútri definovaného časového úseku (*deterministické správanie sa*),
- Všetky požiadavky na RTOS sú spracované v reálnom čase,
- Simultánne vykonávanie viacerých udalostí v reálnom čase,
- Celková absolútna spoľahlivosť systému.

Operačný systém⁵⁹ RTX51 od firmy KEIL môže v prípade potreby reagovať na vzniknutú udalosť v jednotkách μs , alebo *ms* v súlade s požiadavkami programátora. Nemusí teda reagovať na vzniknutú udalosť čo najrýchlejšie⁶⁰, ale len tak rýchlo ako je to požadované, s ohľadom na zachovanie programátorom požadovanej reakčnej doby.

8.3. Terminológia RTOS

Task je samostatná programová jednotka vykonávajúca vopred definovanú činnosť vo virtuálnom prostredí operačného systému. Každá úloha – task je v podstate nekonečná slučka tvorená v jazyku C príkazom **while(1)**, alebo **for(;;)**. Každý task môže RTOS individuálne ovládať. Task pre svoju činnosť v *multitaskovom* prostredí vyžaduje systémové zdroje (*pamäť, časovač, port, sériové rozhranie UART a iné*). Task môže byť samostatný, alebo závislý na iných taskoch (*synchronizácia s inými úlohami*). Všetky tasky sa môžu nachádzať len v jednom z nasledujúcich stavov: **READY**, **BLOCKED**, **RUNNING**, **SLEEPING**.

Multitasking je moderná technológia zabezpečujúca simultánne vykonávanie viacerých úloh na jedinej CPU.

⁵⁹ <http://www.keil.com/pr/article/1253.htm> k dispozícii zdarma.

⁶⁰ Prerušenie sa nesmie stratiť, RTOS ktoré stráca prerušenie nie je použiteľné v praxi a profesionálnej oblasti.

Roundrobin⁶¹ s pridelením konštantných časových kvánt je v podstate kombinovaná metóda ktorá, umožňuje k deterministickému správaniu sa plánovača priradiť ešte plánovanie s riadením priorít. Stručne povedané **timeslicing scheduling = roundrobin**.

Ručný roundrobin odovzdáva vykonávanú úlohu samostatne CPU pre iné tasky v stave **READY** ktoré majú rovnaké priority. Úloha týmto samotná určuje veľkosť časového kvanta nazývaný niekedy ako **kooperatívny multitasking**.

Preempting je termín ktorý vyjadruje násilné ukončenie tasku s nižšou prioritou taskom ktorý má vyššiu prioritu a vykonávaný task je uvedený do stavu **RUNNING** a prerušený do stavu **READY**. Po ukončení tasku s vyššou prioritou je task s nižšou prioritou zo stavu **READY** prepnutý do stavu **RUNNING**.

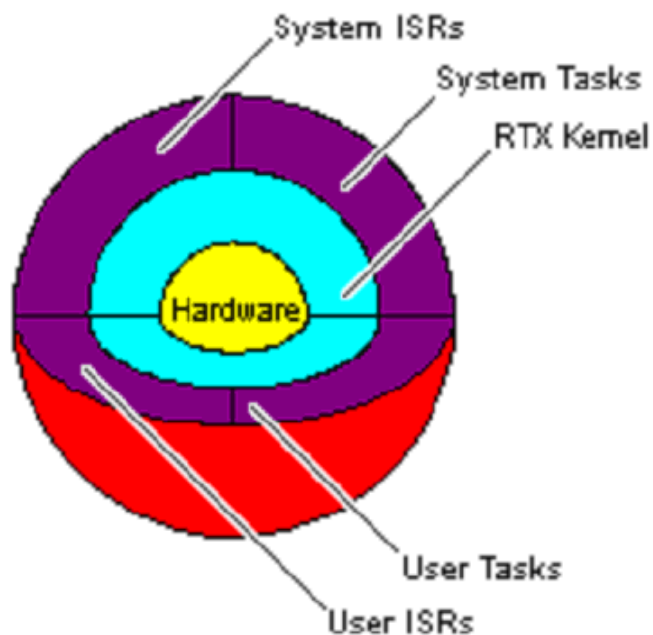
Inverzia priorít je jednou z hlavných úloh plánovača, ktorý má prideliť časové kvantum pre jednotlivé tasky v závislosti na pridelenej prioritě. Ak by však task **A** s vysokou prioritou čakal na uvoľnenie systémového zdroja ktorý môže byť uvoľnený len taskom **B** ktorý má nižšiu prioritu, môžeme vidieť, že task **B** by nemusel byť vzhľadom na vyššiu prioritu tasku **A** prepnutý do stavu **RUNNING**. Preto niektoré RTOS sú vybavené mechanizmom, ktorý umožňuje vzájomnú *inverziu priorít*, ktorá uvedie task **B** z nižšou prioritou na krátky čas do vyššej priority tasku **A**, a zostane v tomto stave pokiaľ nebude task **A** blokovaný taskom **B**. Potom je uvedený task **B** do svojej pôvodnej úrovne priority pred inverziou.

8.4. Funkcia plánovača – scheduler

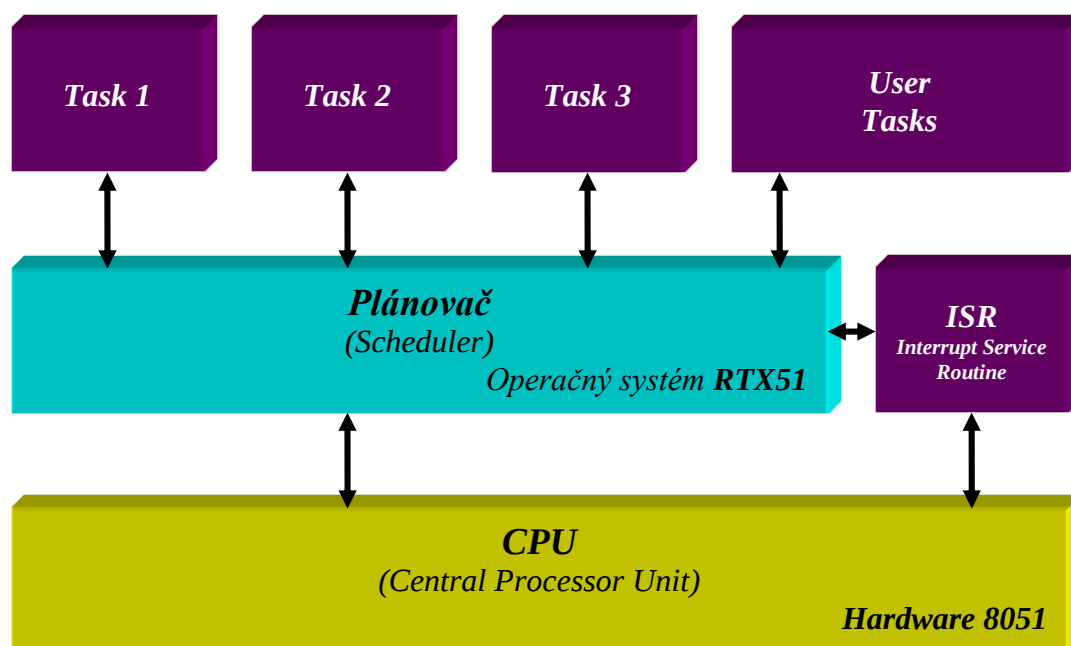
Scheduler, alebo povedané plánovač je najdôležitejším prvkom reálneho operačného systému RTOS. Svojou činnosťou zabezpečuje riadenie úloh tzv. *task management* a zaisťuje, aby CPU bola pridelená úloha s najvyššou prioritou. Všetky tieto informácie sú uložené v riadiacom bloku (*Task Block Control*). Na vlastnostiach plánovača závisia reakčné doby a výkonnosť RTOS.

⁶¹ Pri nastavení **?RTX_TIMESHARING EQU 1** t.j. Round Robin Flag v **RTXSETUP.INI** nie je potrebné používať príkazy **os_wait()** operačného systému RTX51 Full. Operačný systém dokáže prepnúť úlohy aj bez použitia príkazu **os_wait()**.

Obrázok 71, Všeobecný model RTOS



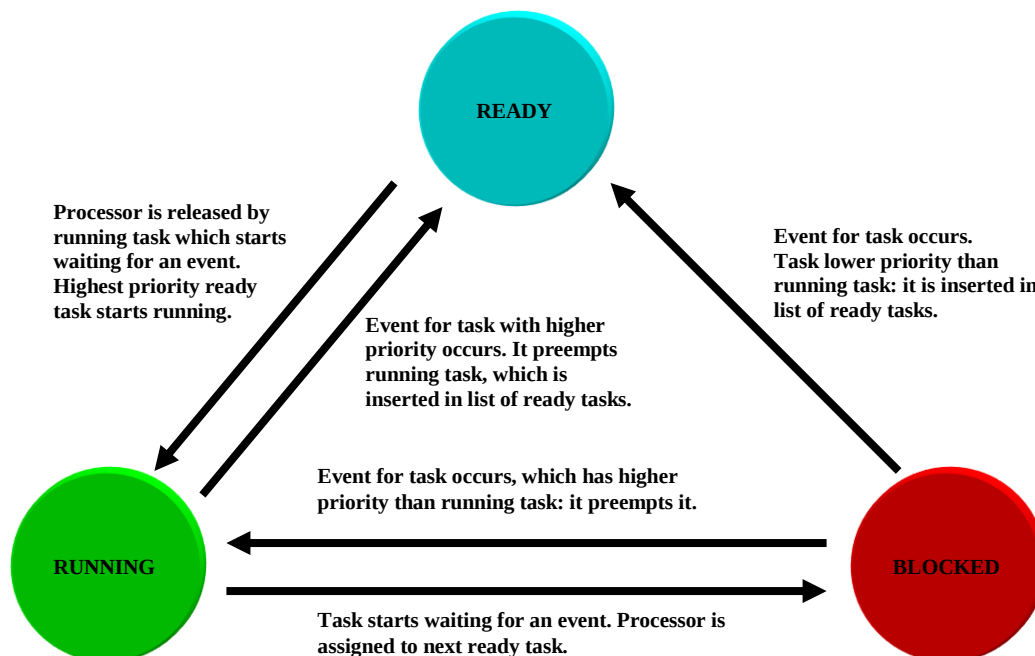
Obrázok 72, Bloková schéma systému reálneho času



Plánovač riadi:

- Správu systémových zdrojov, pamäti, časových kvánt pridelovaných jednotlivým taskom.
- Riadenie prenosu dát medzi jednotlivými úlohami pomocou semaforov, frontmi správ (*Messages Queues*).
- Aktívne riadenie taskov (*pozastavenie, spustenie, prepnutie, nútené prerušenie*).
- Obsluhu prerušenia, na základe ktorého je možné vykonať prepnutie taskov, prerušenie musí byť obslužené a nesmie sa stratiť. RTOS, ktoré stratia prerušenie nie sú schopné prevádzky v aplikáciách v reálnom čase.

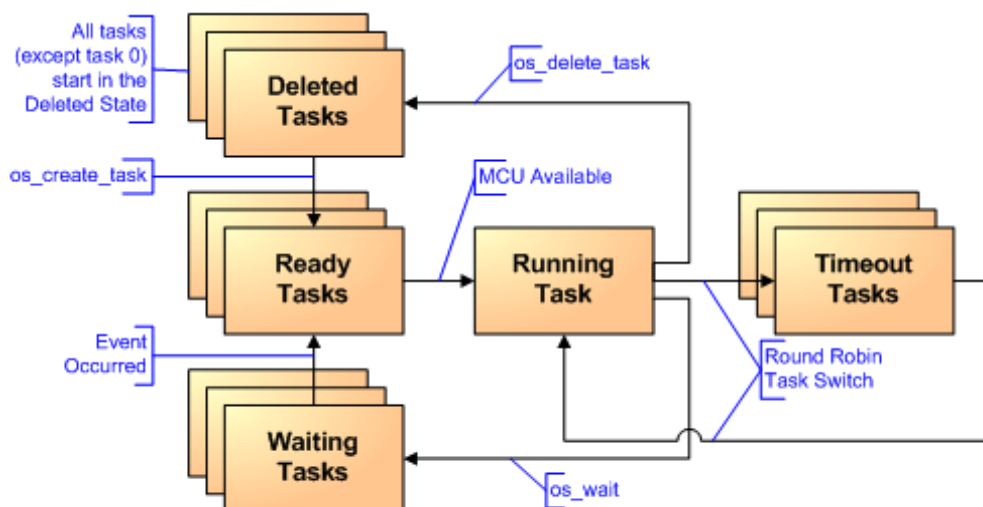
Obrázok 73, Mechanizmus prepínania úloh v RTX51



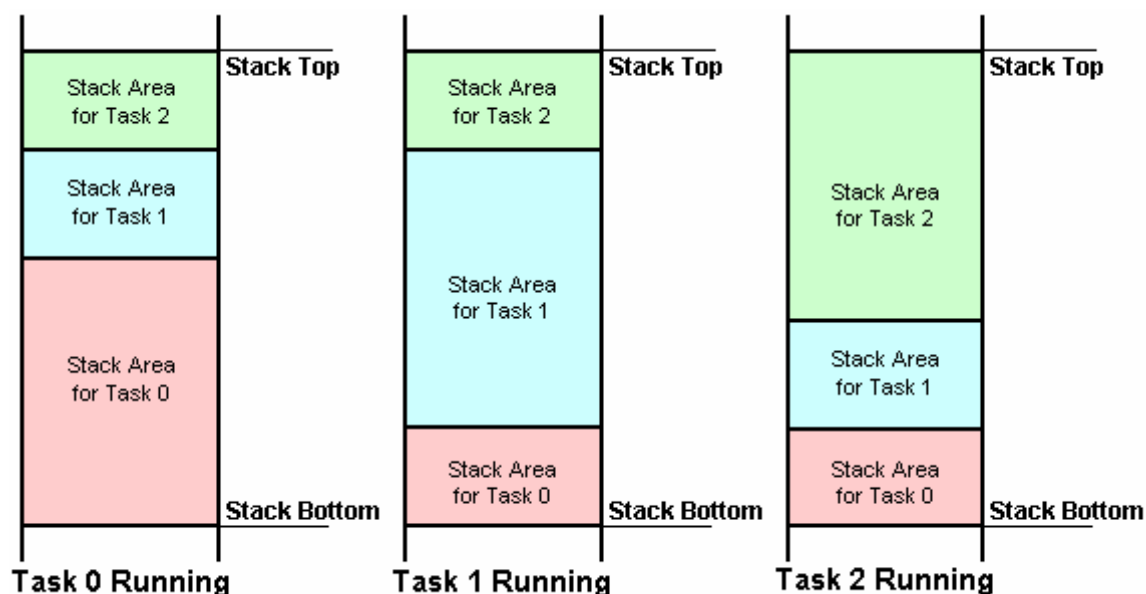
8.5. Spôsob práce plánovača v RTX51

- K prepnutiu úloh dochádza ak je ukončené vykonávanie úlohy s najvyššou prioritou a pokiaľ sa nenachádza v stave **READY**.
- Ak sa v systéme nachádza viac čakajúcich úloh s rovnakou úrovňou priority v stave **READY**, je spustená najdlhšie zaradená čakajúca úloha.
- K prepnutiu úlohy dochádza vtedy, ak nie je splnená prvá podmienka.
- K časovaniu pomocou **round-robin (time-sliced)** dochádza vtedy, ak sú splnené nasledujúce podmienky:
 - a. Musí byť povolené plánovanie **round-robin**.
 - b. Vykonávaná úloha má najnižšiu prioritu 0 a nevykonáva aritmetickú operáciu s reálnym údajovým typom (*float*).
 - c. Minimálne jedna úloha s prioritou 0 musí byť v stave **READY**.
 - d. Musia byť nastavené časové kvantum funkciou **os_set_slice(Number)**.

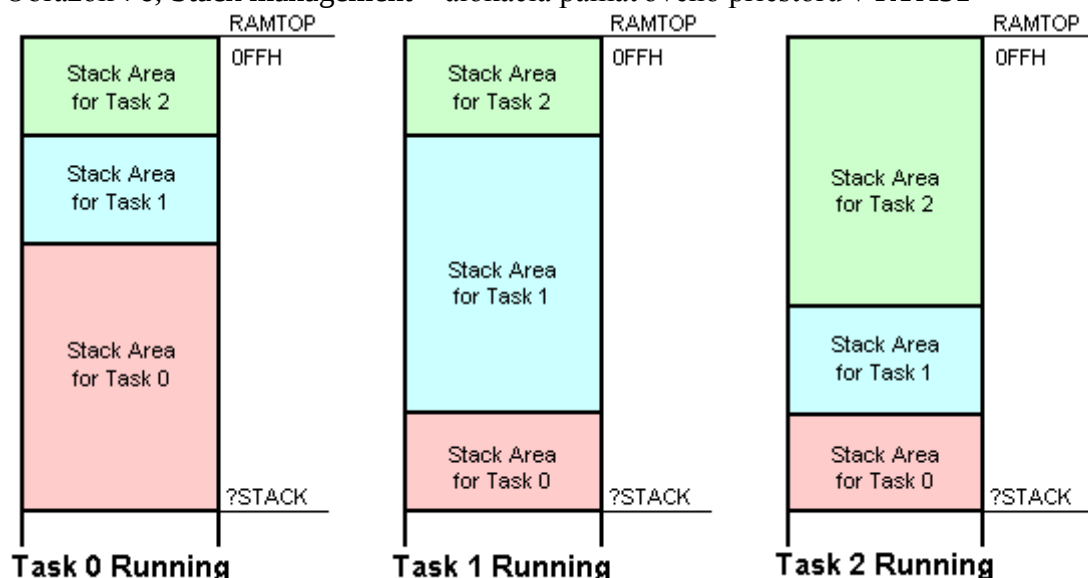
Obrázok 74, Task management – prepínanie úloh v RTX166



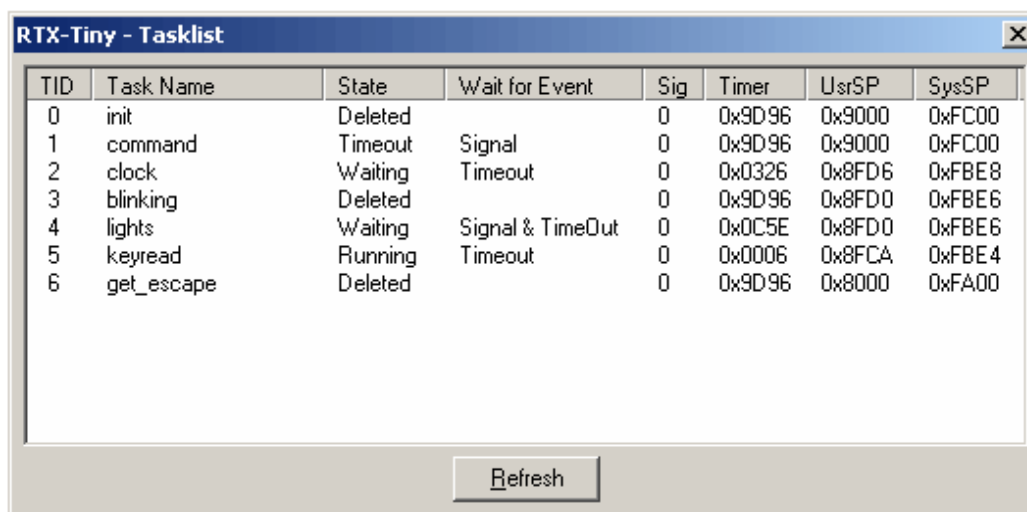
Obrázok 75, Stack management – alokácia pamäťového priestoru v RTX166



Obrázok 76, Stack management – alokácia pamäťového priestoru v RTX51

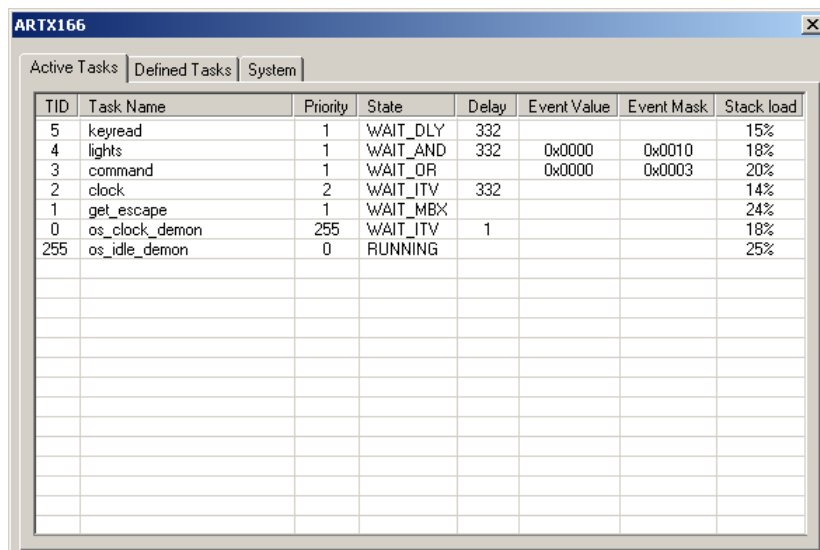


Obrázok 77, Debugging – Stav úloh a sys. prostriedkov RTX166 Tiny a AR166



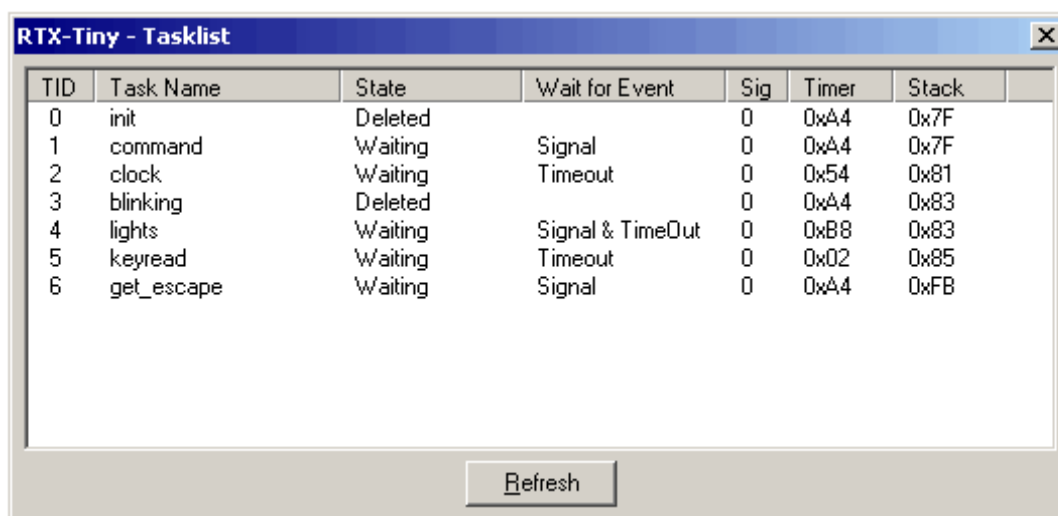
TID	Task Name	State	Wait for Event	Sig	Timer	UsrSP	SysSP
0	init	Deleted		0	0x9D96	0x9000	0xFC00
1	command	Timeout	Signal	0	0x9D96	0x9000	0xFC00
2	clock	Waiting	Timeout	0	0x0326	0x8FD6	0xFBE8
3	blinking	Deleted		0	0x9D96	0x8FD0	0xFBE6
4	lights	Waiting	Signal & TimeOut	0	0x0C5E	0x8FD0	0xFBE6
5	keyread	Running	Timeout	0	0x0006	0x8FCA	0xFBE4
6	get_escape	Deleted		0	0x9D96	0x8000	0xFA00

Refresh



TID	Task Name	Priority	State	Delay	Event Value	Event Mask	Stack load
5	keyread	1	WAIT_DLY	332			15%
4	lights	1	WAIT_AND	332	0x0000	0x0010	18%
3	command	1	WAIT_OR		0x0000	0x0003	20%
2	clock	2	WAIT_ITV	332			14%
1	get_escape	1	WAIT_MBX				24%
0	os_clock_demon	255	WAIT_ITV	1			18%
255	os_idle_demon	0	RUNNING				25%

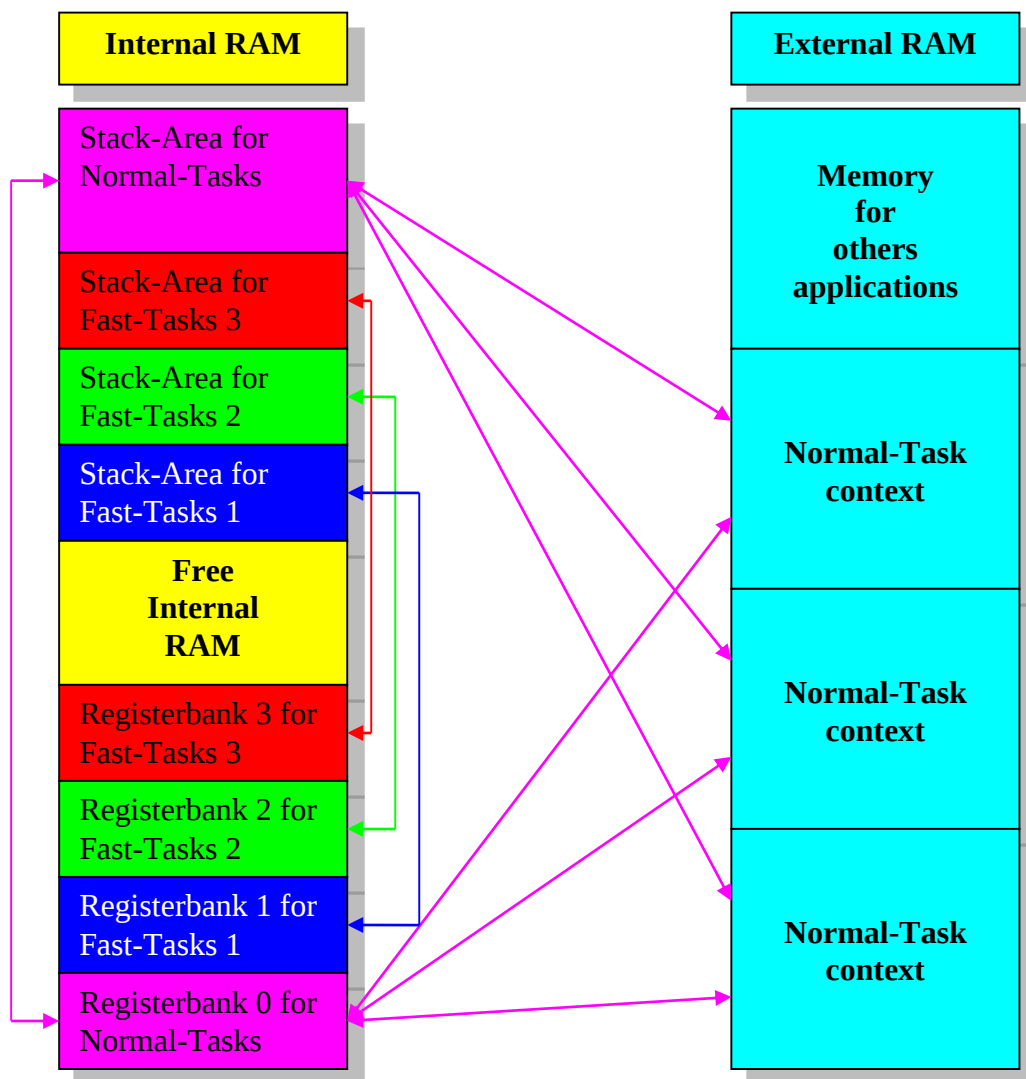
Obrázok 78, Debugging – Stav sputených úloh a systémových prostriedkov RTX51 Tiny



TID	Task Name	State	Wait for Event	Sig	Timer	Stack
0	init	Deleted		0	0xA4	0x7F
1	command	Waiting	Signal	0	0xA4	0x7F
2	clock	Waiting	Timeout	0	0x54	0x81
3	blinking	Deleted		0	0xA4	0x83
4	lights	Waiting	Signal & TimeOut	0	0xB8	0x83
5	keyread	Waiting	Timeout	0	0x02	0x85
6	get_escape	Waiting	Signal	0	0xA4	0xFB

Refresh

Obrázok 79, Typy úloh a alokácia pamäťového priestoru v RTX51 Full



Obrázok 79, Typy úloh a alokácia pamäťového priestoru v RTX51 Full farebne znázorňuje stratégiu používania jednotlivých registrových bánk RB0 až RB3 s RTX51 Full. Pre úlohy s prioritou 0 až 2 (fialová farba) je používaná výhradne banka registrov RB0. Všetky úlohy štandardne zdieľajú banku registrov RB0. Banky registrov RB1 a RB3 môže využiť len úloha deklarovaná ako fast-task s najvyššou prioritou 3. Každá úloha s najvyššou prioritou musí mať vlastnú banku registrov a nie je možné zdieľať viacerými úlohami rovnakú banku registrov. Ak vyššie uvedené nebude dodržané program nebude korektne pracovať.

8.6. Vzájomná komunikácia medzi úlohami v RTOS

Komunikácia v RTX51 môže byť vykonávaná pomocou:

- Fronty správ (*message queue*).
- Semaforom.
- Binárnym semaforom.
- Prírastkovým semaforom.

Fronta správ je v podstate druh paketu (*bloku dát*), ktorý je odovzdávaný iným úlohám. Task môže oznámiť očakávanie správy a okamžite prechádza do stavu BLOCKED. Ak je prijatá očakávaná správa a má práve najvyššiu prioritu tak je prerušený práve vykonávaný task a riadenie je odovzdané tasku očakávajúcemu správu. Správa môže byť v závislosti na výrobcovi RTOS pevnú, alebo premenlivú dĺžku. RTX51 umožňuje definovanie obidvoch typov správ.

Semafor umožňuje svojou funkciou zdieľanie systémových zdrojov viacerým úlohám bez ich vzájomnej kolízie. Výborným príkladom je súčasný prístup k sériovému rozhraniu UART dvomi úlohami. V prípade ak jedna úloha vysiela dáta cez sériové rozhranie, musí byť druhá úloha blokováná, aby nevysielala na rovnaký sériový kanál. V prípade plne duplexného sériového kanála mikroprocesora 8051 je možné, aby jedna úloha vysiela na rozhranie a druhá úloha prijímala údaje zo sériového rozhrania. Ale súčasná funkcia vysielania a príjmu dát viacerými úlohami je blokováná použitím semafora.

Binárny semafor⁶² na rozdiel od predchádzajúceho semafora umožňuje rezervovať systémové zdroje CPU len pre jeden task. Môžu teda nadobúdať stav TRUE, alebo FALSE.

Prírastkový semafor oproti vyššie uvedených semaforov majú možnosť definovať len určitý prípustný počet súčasne prístupujúcich taskov k zdieľaným perifériám CPU. Každý task vyžadujúci použitie semafora zníži stav semafora o 1. Každý použitý semafor zasa zvýši túto hodnotu o 1. V prípade ak je semafor použitý maximálnym počtom taskov, prechádza do 0 a ďalší task žiadajúci semafor prechádza do stavu BLOCKED, pokiaľ iný task požadovaný semafor neuvolní.

⁶² Použitý v RTX51 Full

9 Systémové požiadavky na operačný systém RTX51

9.1. RTX51 Tiny

- Maximálne je možné deklarovať 16 úloh.
- V podstate je to časť funkcií verzie RTX51 Full.
- Vhodná pre jedno čipové mikropočítače bez externej pamäti dát a programu.
- Nepodporuje preemptívny scheduler – plánovač.
- Nepodporuje vzájomnú komunikáciu medzi úlohami pomocou správ.
- Nie je možné alokovať spoločne zdieľané dátové oblasti v RAM.

Hardwarové požiadavky na RTX51 Tiny verzie 2.02:

- 7 Byte pamäti dát RAM.
- 900 Byte v pamäti programu CODE.
- RTX51 Tiny⁶³ vyžaduje 3*[počet úloh] nepriamo adresovateľnej pamäti IDATA.

9.2. RTX51 Full

- Maximálne je možné deklarovať 256 úloh, z toho môže byť 19 úloh aktívnych.
- Štandardné úlohy môžu pracovať s prioritou 0 až 2, (*max. 16 úloh*).
- Tzv. Fast-úlohy pracujú s najvyššou prioritou 3, (*max. 3 úlohy*).
- Umožňuje round-robin a preemptívny scheduler (len RTX51 Full⁶⁴).
- Poskytuje až 4 úrovne priorít číslovaných od 0 po 3.
- Pracuje paralelne s prerušovacím systémom 8051.
- Využíva MAILBOX na prenos signálov a správ medzi úlohami (*max. 8*).
- Použitie max. 8 binárnych semaforov pre zdieľanie periférií v 8051.
- Je možné alokovať max. 16 spoločných pamäťových priestorov.
- Úloha môže aktívne čakať na prerušenie, časový interval, signál, alebo správu od inej úlohy, alebo prerušenia.
- Maximálne oneskorenie na prerušenie je menej ako 50 cyklov mikroprocesora.
- Možnosť zvoliť použitie časovača T0, T1, T2 pre jadro RTX51.
- Nastaviteľný systémový časovač od 1000 do 40000 cyklov.
- Existencia rôznych pamäťových modelov⁶⁵ SMALL, COMPACT a LARGE.

⁶³ RTX51 Tiny je po menšej úprave použiteľný aj pre malé mikropočítače napr. AT89C4051 s minimálnym množstvom 4kB pamäte programu.

⁶⁴ RTX51 Full je robustný RTOS pre náročné aplikácie. Vyžaduje min. 1kB externej RAM.

⁶⁵ Záleží na použítom mikroprocesore a jeho internej alebo externej pamäti. Pri použití funkcie printf() môže pri modeli SMALL vzniknúť nedostatok internej pamäti RAM, pretože funkcia potrebuje k svojej činnosti alokovať

Hardwarové požiadavky na RTX51 Full 7.0:

- 40 až 46 Byte pamäti dát DATA.
- 900 Byte v pamäti programu CODE.
- 20 až 200 Byte v pamäti dát IDATA.
- min. 650 Byte v externej pamäti dát XDATA.
- 6 až 8 kB pamäti programu CODE.

9.3. Pamäťový priestor RTX51

Štandardná harwardská architektúra mikroprocesora 8051 je schopná adresovať 64kB pamäti programu a dát. V niektorých aplikáciách je vyžadované väčšie množstvo pamäti dát, alebo programu a rieši sa to pridaním niekoľkých adresných vodičov k existujúcim, ktoré vytvoria v podstate virtuálny pamäťový priestor. Tento spôsob správy pamäti je nazývaný ako **xBANKING** a umožňuje rozšíriť pamäťový priestor až na 16MB.⁶⁶

Jeho funkciu je možné vysvetliť ako *n* segmentov pamäti RAM, alebo ROM pripojených „paralelne“ k mikroprocesoru 8051, pričom generovanie výberového signálu pre konkrétnu pamäť je zabezpečená výberovými signálmi ***PSEN***, ***ALE***, ***RD***, ***WR*** a doplnujúcimi adresnými vodičmi na porte **P1** potrebných pre **xBANKING**. Linker **BL51** vytvorí *n* úsekov pamäti programu ku ktorým priradí ešte nevyhnutnú obsluhu pamäťového priestoru programu a dát.

Maximálne môže RTX51 Tiny⁶⁷ a Full obsluhovať až 64 bánk programu a dát (t.j. 4MB) pričom každá pamäť môže mať maximálne 2¹⁶ byte t.j. 64kB. Pri použití moderných derivátov mikroprocesorov rady 51 je možné používať až 256 bánk programu a dát čo predstavuje (16MB) pamäťového priestoru pre aplikácie.

Obrázok 80, Nastavenie premennej RAMTOP pre RTX51 Tiny

```
83 ;  
84 ; Define the highest RAM address used for CPU stack  
85 RAMTOP EQU 07FH ; default is address (256-1)  
86 ;  
87 FREE_STACK EQU 20h ; default is 20 bytes free space on stack  
88 ; ; the value 0 disables stack checking
```

cca. 40Byte, ktoré nikde nevypisuje ako alokované a program potom vykonáva skok do častí pamäte programu kde sa nenachádza žiadny program.

⁶⁶ Túto pamäť môžeme obsluhovať pomocou portu P1, na ktorom dochádza k výberu aktívnej *n*-tej fyzickej pamäti RAM, alebo ROM. Pamäť RAM je už potom považovaná za xBank RAM.

⁶⁷ Pre jednočipové mikroprocesory ktoré majú len 128B pamäti RAM je nutné nastaviť v conf_tny.a51 RAMTOP na hodnotu 07Fh, ináč RTX51 Tiny nefunguje správne. Pre procesory 8051 ktoré majú 256B internej pamäti RAM je RAMTOP nastavená na hodnotu 0FFh.

10 Operačný systém reálneho času RTX51

10.1. Vzájomná komunikácia a synchronizácia medzi úlohami v RTX51

RTX51 poskytuje tri spôsoby ktoré povoľujú individuálnej úlohe synchronizovať sa a komunikovať s ďalšími úlohami:

- **SIGNALS** sú v podstate riadiace signály neobsahujúce informácie o zdrojovej úlohe a ich úlohou je synchronizovať vykonávanie jednotlivých úloh.
- **MESSAGES** sú dáta s pevnou, alebo premenlivou dĺžkou, ktoré sú prenášané cez tzv. *mailbox* a slúžia na vzájomnú výmenu dát medzi úlohami. Každý mailbox je schopný prijať až 8 správ⁶⁸ v poradí ako boli prijaté.
- **SEMAPHORES** sú jednoduchým mechanizmom, ktorý je schopný zdieľať systémové zdroje CPU. Semaforey sa používajú vtedy, ak minimálne dve úlohy používajú súčasne jedno zariadenie⁶⁹. Vhodným príkladom je napríklad sériové RS232, CAN, SPI, alebo I²C rozhranie. Vzájomne sa teda blokuje prístup druhej úlohy k rozhraniu ak s ním pracuje prvá úloha. V RTX51 Full je možné použiť len binárne semaforey.

10.2. Mechanizmus prepínania úloh v RTX51

RTX51 rozoznáva 4 základné stavy:

- **READY** Všetky úlohy ktoré nie sú v stave **BLOCKED** alebo **RUNNING** a sú pripravené k spusteniu sú v stave **READY**.
- **RUNNING** V stave **RUNNING** sa nachádza úloha ktorú práve vykonáva CPU.
- **BLOCKED** V tomto stave sa nachádza úloha ktorá aktívne čaká na výskyt udalosti (*timeout, message, semaphore, alebo signal*).
- **SLEEPING** Tento stav nadobudne úloha ktorá je síce deklarovaná, ale nebola ešte spustená. Zároveň tento stav nadobudne úloha ktorá bola operačným systémom ukončená. Úloha sa môže nachádzať len v jednom z vyššie uvedených stavov a zmena z jedného do druhého stavu je charakterizovaná ako skoková.

⁶⁸ Správa t.j. **Messages** je premenná typu **int**, alebo **unsigned int**. Niekedy je zbytočne a nezmyselne vypisované hlásenie RTX51 o nesprávnom použití údajovom type. Pritom sú v podstate tieto typy v RTX51 rovnocenné. V prípade, že bude vypisované hlásenie o typovej nekompatibilitate, je potrebné vyskúšať oba údajové typy.

⁶⁹ Ako príklad môžeme považovať funkciu **printf()**, ktorá je použitá v dvoch rôznych task-och. Z pohľadu RTX51 sa ale jedná o „súčasný“ prístup do rovnakej procedúry. Pokiaľ táto funkcia nie je deklarovaná ako reentrant-ná, dôjde k poškodeniu environment-u funkcie **printf()** a tým analogicky aj k nefunkčnosti funkcie.

10.3. Riadenie úloh pomocou udalostí v RTX51

Úloha môže aktívne čakať na výskyt jednej udalosti, alebo kombinácie udalostí ktoré sa vyskytnú v priebehu vykonávania programu. Pokiaľ jedna úloha čaká na udalosť, ostatné úlohy môžu byť v stave **READY** a môžu byť prepnuté do stavu **RUNNING**.

Riadenie pomocou nižšie uvedených udalostí je možné pomocou systémovej funkcie RTX51 **os_wait(K_EVENT,TIME,NULL)**⁷⁰, kde parameter **K_EVENT** umožňuje definovať typ očakávanej udalosti.

Zoznam udalostí pre funkciu **os_wait()**:

- **Semaphore**⁷¹ Len RTX51 Full, používa sa pre obsluhu zdieľaných systémových zdrojov napr. volanie funkcie **printf()** z viacerých taskov.
- **Signal** Používa sa na vzájomnú synchronizáciu jednotlivých úloh.
- **Interrupt**⁷² Len RTX51 FULL, úloha môže čakať na hardwarové prerušenie.
- **Message**⁷³ Len RTX51 FULL, používa sa pre výmenu správ medzi úloh.
- **Time-out** Preruší vykonávanie úlohy na definovaný počet hodinových cyklov systémového časovača RTX51 a odovzdá riadenie ďalšej úlohe – task-u.
- **Interval** Len RTX51 Tiny⁷⁴, je ekvivalentný *timeout-u* ale systémový časovač nie je po pretečení resetovaný a tým generuje veľmi presné časové úseky v programe.

V programe je možné využiť aj kombináciu vyššie uvedených udalostí v jednom príkaze a to napríklad: **os_wait(K_TMO+K_SIG,100,0)**, kde RTX51 očakáva príjem signálu od inej úlohy po dobu 100 cyklov systémového časovača. **K_TMO** je v tomto prípade „zbytočná“, pretože **K_SIG** je nastavená tak, že po 100 cykloch časovača sa ukončuje automaticky príjem čakania na signál a správa sa ukončuje akoby sa aktivoval „timeout“ s udalosťou **K_TMO**. Ak použijeme **os_wait(K_IVL+K_SIG,100,0)** a preruší sa čakanie na udalosť iným typom udalosti

⁷⁰ RTX51 Full oproti RTX51 Tiny neobsahuje **os_task_switch()** a preto sa používa funkcia **os_wait(K_TMO,0,0)**.

⁷¹ Nie je možné kombinovať v jednom task-u súčasný príjem správy z mailbox-u **MSG_EVENT** (0-7) a token-u **SEM_EVENT** semaforu (8-15).

⁷² Prerušenie na túto udalosť je vhodné nastaviť lepšie pomocou funkcie **os_attach_interrupt(n)**, ako pomocou **os_enable_isr(n)**, kde **n = (0..31)** pretože funkcia **os_attach_interrupt(n)** nastaví kompletne príslušný register IE a nepotrebujeme napísať obsluhu t.j. handler pre dané prerušenie **n**.

⁷³ Odoslanie správy sa vykonáva pomocou funkcie **os_send_message(mbx,var,100)** odovzdáva obsah premennej **var** príslušnému MailBox-u **mbx** a čaká na doručenie 100 cyklov. Prijatie správy je v tvare **os_wait(K_TMO+K_SIG+K_MBX+mbx,200,&vars)**, kde sa obsah správy prijateľ cez mailbox **mbx** uloží na adresu premennej **&vars**.

⁷⁴ Nezodpovedá to úplne skutočnosti ktorá je uvedená v príručke programátora, pretože tento typ udalosti v **os_wait(K_IVL,100,NULL)** funguje aj v RTX51 Full a umožňuje generovanie veľmi presných časových intervalov ktoré sú odvodené od hardware systémového časovača T0, T1, alebo T2.

ako je „timeout“ je nutné použiť procedúru **os_reset_interval(100)**, ktorá vykoná opätovné nastavenie hardware časovača RTX51. Obrázok 81 ukazuje neštandardné použitie udalosti typu „timeout“, ktorá generuje presné časové intervaly a zároveň očakáva udalosť typu „signal“. V prípade ukončenia čakania pomocou udalosti „signal“ je podľa príručky RTX51 nevhnutné ukončiť pomocou funkcie **os_reset_interval(100)** čakanie, aby nedošlo ešte opätovne k zbytočnému vyvolaniu udalosti „timeout“. Pri krokovani programu sme zistili, že aj napriek použitému príkazu **os_reset_interval(100)** nedôjde k resetu hardware časovača a vyvolá sa zbytočne ešte navyše bezdôvodne „timeout“ udalosť. Obrázok 82 zobrazuje rovnaký task v ktorom je použitý **os_wait(K_TMO+K_SIG,50,0)** ktorý pracuje správne, lebo nepotrebuje resetovať hardware časovača RTX51 a prijatím udalosti typu „signal“ ukončí korektne task.

Obrázok 81, Nesprávne ošetrenie udalosti typu "interval"

```
void Clock(void) _task_ CLOCK
{
    while(1)
    {
        switch(os_wait(K_IVL|K_SIG,100,NULL))
        {
            case TMO_EVENT : i++; break;
            case SIG_EVENT : Counter++; os_reset_interval(100); break;
        }
    }
}
```

Obrázok 82, Správne ošetrenie udalosti typu "timeout"

```
void System(void) _task_ SYSTEM
{
    os_create_task(CLOCK);
    os_create_task(TICKS);
    while(1)
    {
        switch(os_wait(K_TMO|K_SIG,50,NULL))
        {
            case TMO_EVENT : break;
            case SIG_EVENT : break;
        }
    }
}
```

Príklad využitia `os_wait(event_selector,timeout,0)` v kombinácii s inými udalosťami

c:\Omega\data\Dropbox\Záloha\C51\i2c_dsw\i2c_DSW.c

```
86 void Adc (void) _task_ ADC _priority_ 3
87 {
88     signed char RtxCompletion;
89     os_attach_interrupt (10);
90     while (1)
91     {
92         Watchdog;
93         switch (RtxCompletion=os_wait (K_TMO+K_INT+K_SIG, 10, NULL))
94         {
95             case SIG_EVENT : Voltage=ADC_Conversion (2); break;
96             case TMO_EVENT : ADCON |=( ADCS+2);
97             case INT_EVENT : ADCON ^=ADCI; Voltage=Conversion (); break;
98         }
99     }
100 }
```

10.4. Prerušovací podsystem RTX51

Obsluha prerušenia je jednou z hlavných a najdôležitejších úloh, ktoré musí RTX51 výborne ovládať. Mnoho operačných systémov práve stroskotá pri obsluhu viacnásobného príjmu prerušení v reálnom čase. **Prerušenie sa v žiadnom prípade nesmie stratiť!**

RTX51 podporuje dve rozdielne kategórie úloh:

1. Standard-task:

- Vyžadujú viac času na prepnutie medzi úlohami.
- Všetky úlohy zdieľajú spoločnú banku registrov **RB0** a zásobník **STACK**.
- Sú prerušiteľné fast-task úlohami a štandardnými prerušeniami z C51.
- Môžu sa vzájomne prerušovať podľa vopred deklarovaných pravidiel.
- Maximálne 16^{75} aktívnych úloh v programe, (256 neaktívnych).

2. Fast-task:

- Používané pre rýchlu reakciu na výskyt udalosti.
- Úlohy majú najvyššiu úroveň priority 3.
- Obsahujú vlastný separátny (oddelený) zásobník **STACK**.
- Používajú fyzicky oddelené banky registrov **RB1 až RB3** pre fast-tasks⁷⁶.
- Môžu byť prerušené len výskytom štandardného prerušenia HW⁷⁷.
- Maximálne môžu byť v systéme definované 3 fast-task úlohy.
- Možnosť využiť pre prerušenie **PFI**⁷⁸ (Power Fail Interrupt)

⁷⁵ Súčasne je možné spustiť v RTX81 Full 16+3 t.j. 19 taskov.

⁷⁶ Prepnutie úlohy s prioritou 3 trvá max. 50 cyklov mikroprocesora, musí byť použitá reg. banka 3 pomocou direktívy **#pragma REGISTERBANK (3)** ako to ukazuje vyššie uvedený príklad. Štandardné úlohy s prioritou **0 až 2** využívajú banku registrov **0**, fast-task úlohy vyžadujú každá samostatne použitú banku registrov (1 až 3).

⁷⁷ Niektorého prerušenia od hardware 8051.

⁷⁸ Niektoré procesory PFI nepodporujú, viď manuál C51 a Cx51 na www.keil.com.

Príklad úlohy typu Fast-Task:

```
84  #pragma REGISTERBANK (3)
85
86  void Adc (void) _task_ ADC _priority_ 3
87  {
88      signed char RtxCompletion;
89      os_attach_interrupt (10);
90      while (1)
91      {
92          Watchdog;
93          switch (RtxCompletion=os_wait (K_TMO+K_INT+K_SIG,10,NULL))
94          {
95              case SIG_EVENT : //Voltage=ADC_Conversion(2); break;
96              case TMO_EVENT : ADCON |=( ADCS+2);
97              case INT_EVENT : ADCON ^=ADCI; Voltage =Conversion (); break;
98          }
99      }
100 }
```

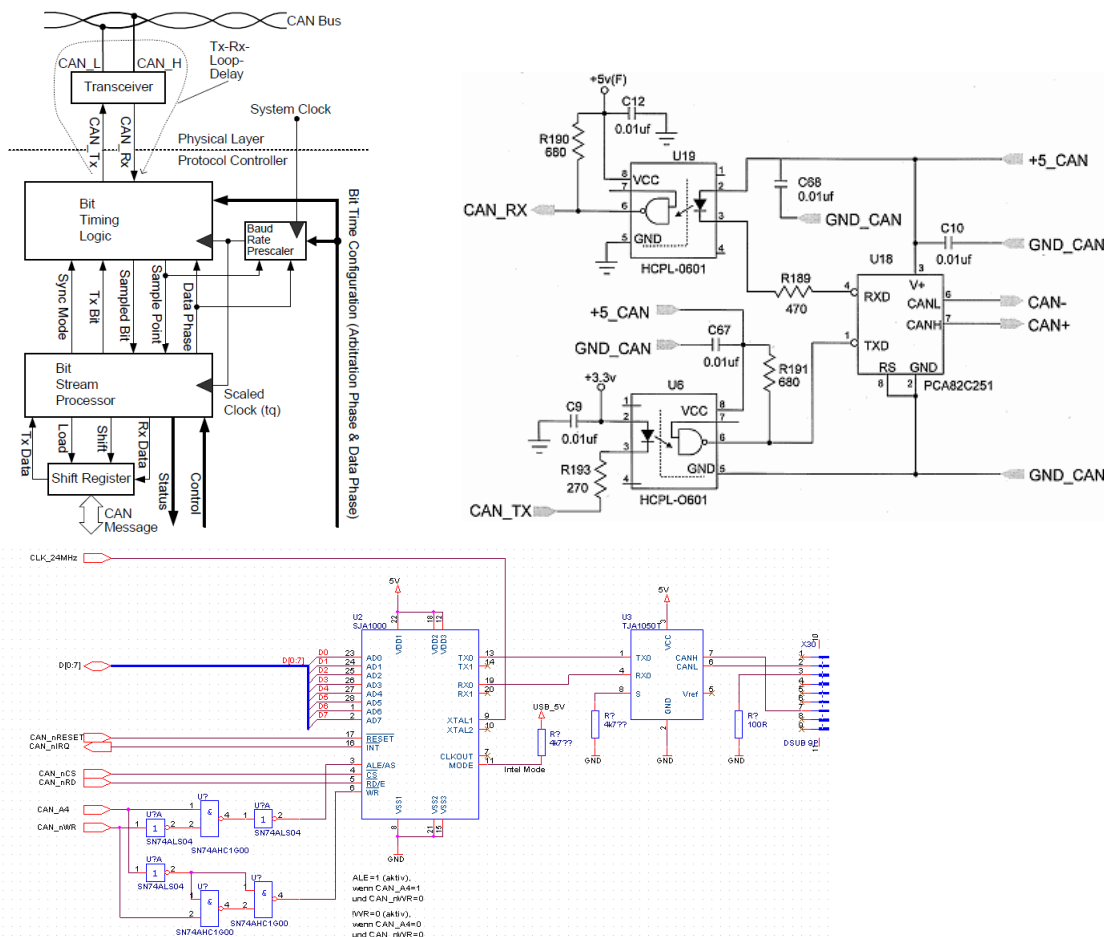
11 Komunikácia RTX51 s periférnym rozhraním CAN

Komunikačné rozhranie CAN definovala a štandardizovala firma BOSCH v roku 1986. Využitie sa našlo hlavne v automobilovom priemysle, kde sa vďaka nej znížila hmotnosť použitých vodičov na komunikáciu s jednotlivými komponentami umiestnených vo vozidle.

11.1. Vlastnosti CAN

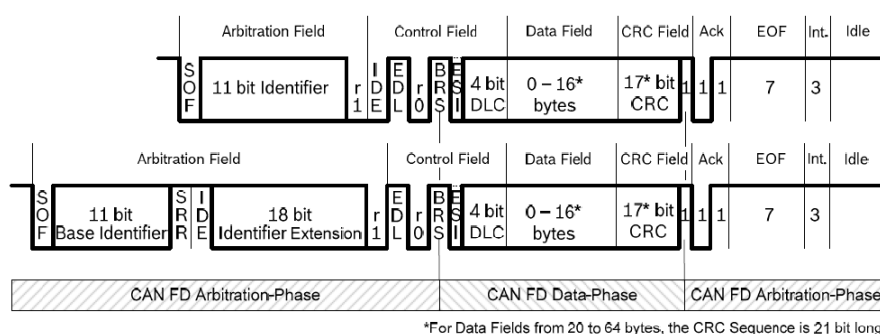
- ✓ sériový typ komunikácie pre aplikácie v reálnom čase,
- ✓ rýchlosť prenosu dát až do $1,0 \text{ Mbs}^{-1}$ (terminátor 124Ω na oboch koncoch vedenia),
- ✓ extrémne vysoká odolnosť voči chybám pri prenose dát,
- ✓ predurčená na použitie v automobile,
- ✓ použitie v priemyselnej automatizácii a pri riadení aplikácií,
- ✓ definovaný priemyselný štandard **ISO11898**⁷⁹.

Obrázok 83, Komunikácia CAN s fyzickou vrstvou CAN Bus



79 http://www.semiconductors.bosch.de/media/pdf/canliteratur/can_fd.pdf

Obrázok 84, Formát CAN správ



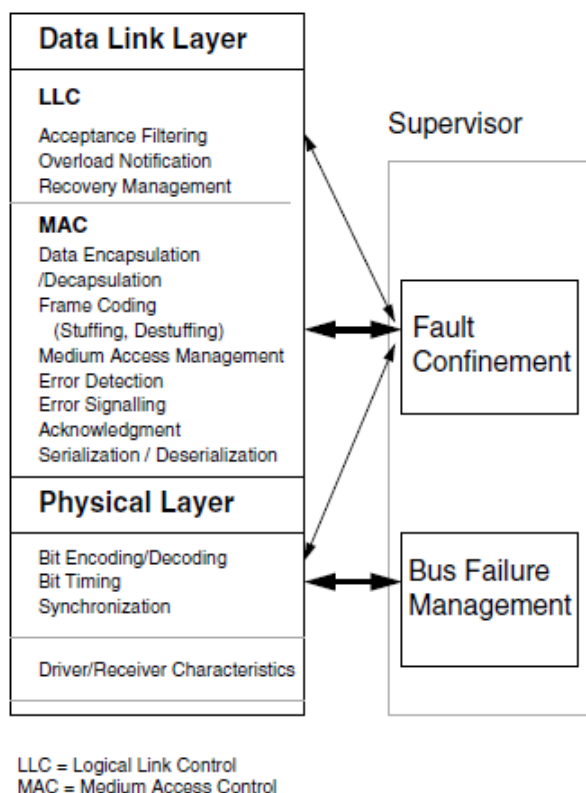
11.2. Komunikačný model CAN

Komunikačný model rozhrania CAN je veľmi podobný komunikačnému modelu TCP/IP v sieti Ethernet. Maximálna dĺžka medzi dvomi komunikujúcimi zariadeniami v CAN na ktorej je schopná komunikovať rýchlosťou $1,0 \text{ Mbs}^{-1}$ je cca. 40m, rýchlosťou 125kbps až na vzdialenosť 500m.

Každý uzol vyžaduje:

- ✓ hostiteľský procesor,
- ✓ CAN radič,
- ✓ vysielateľ (TxD, RxD).

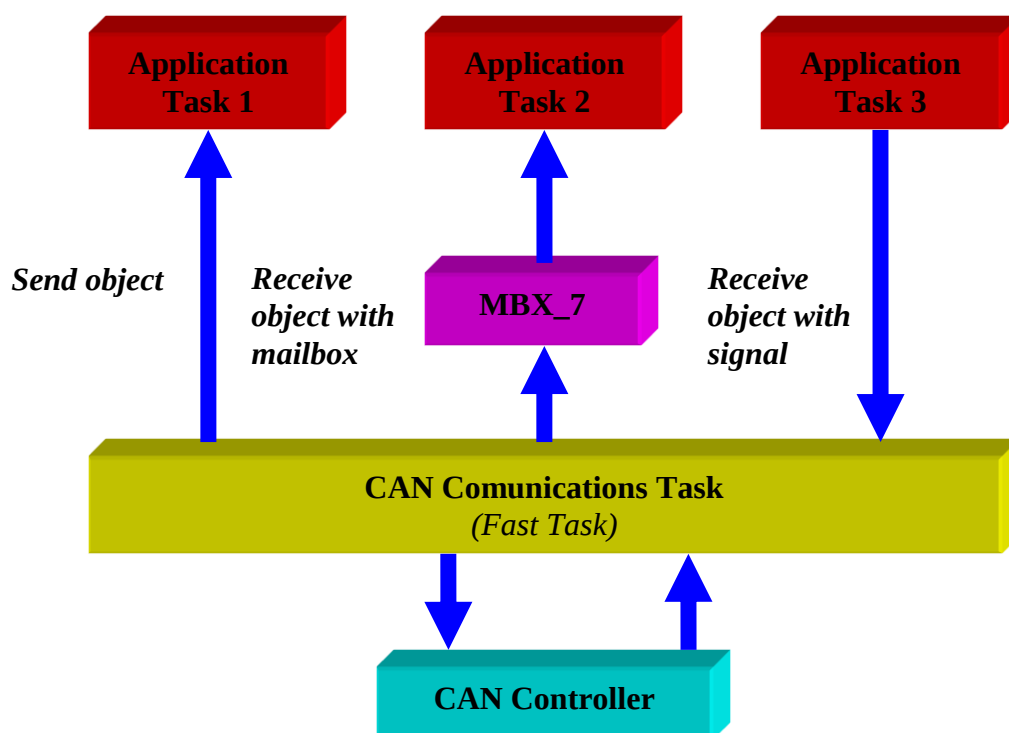
Obrázok 85, Základný koncept CAN



Tabuľka 8, Podpora⁸⁰ RTOS od firmy KEIL pre rozhranie CAN

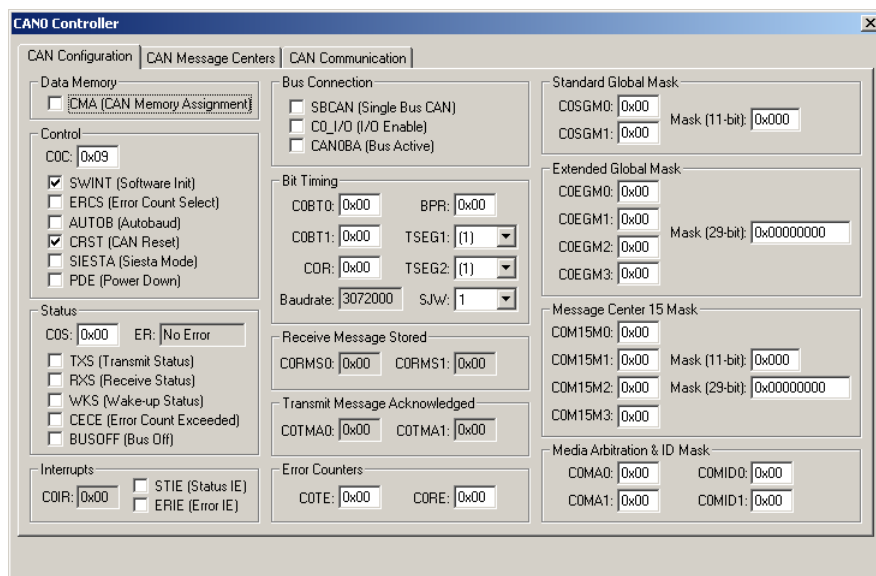
Part Number	RTX51 Real-Time Kernel	RTX166 Real-Time Kernel
CAN Devices Supported		
Infineon 81C90/81C91 (Stand-alone CAN controller)	✓	
Infineon C505C (8051-based Micro with CAN)	✓	
Infineon C515C (8051-based Micro with CAN)	✓	
Infineon C167CR (C16x-based Micro with CAN)		✓
Infineon C164CI (C16x-based Micro with CAN)		✓
Intel 82526/82527 (Stand-alone CAN controller)	✓	
Philips 82C200 (Stand-alone CAN controller)	✓	
Philips 8xC592 (8051-based Micro with CAN)	✓	
Philips 8xC598 (8051-based Micro with CAN)	✓	

Obrázok 86, Spôsob komunikácie aplikácie RTOS a rozhraním CAN

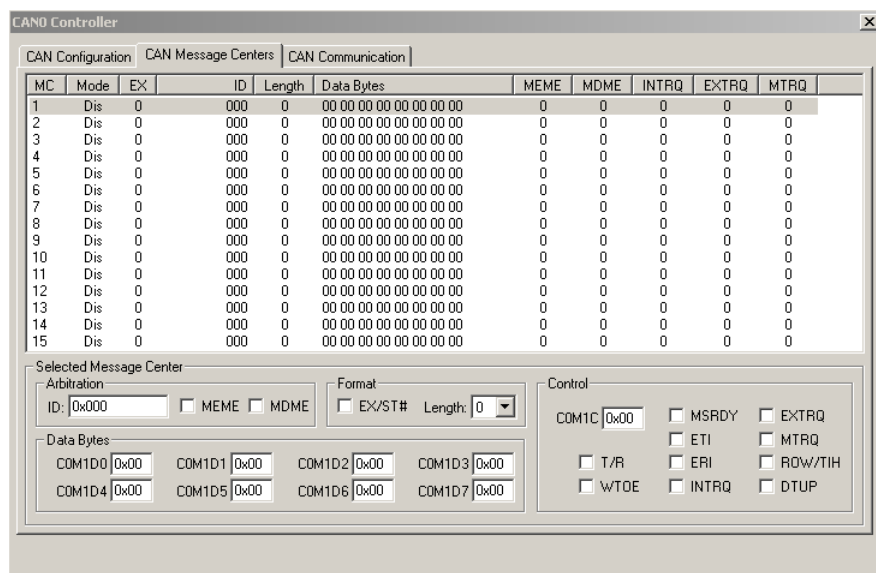


⁸⁰ http://www.keil.com/uvision/db_sim_prf_can.asp, RTX51 Full verzie 7.0 podporuje navyše mimoriadne populárny obvod SJA1000T od firmy Philips Semiconductor. Bližšie informácie o obvode SJA1000T nájdete na http://www.nxp.com/documents/application_note/AN97076.pdf.

Obrázok 87, CAN Controller v μ Vision4



Obrázok 88, CAN Message Center v μ Vision4



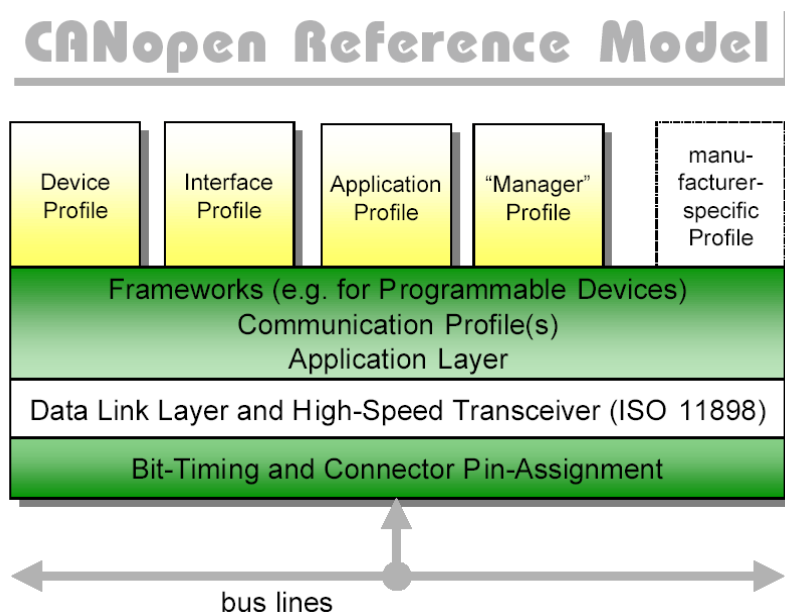
Obrázok 89, Generovanie⁸¹ CAN správ v prostredí μ Vision4

```
SIGNAL void sendCAN (float secs) {
    while (1) {
        CANOID = 0x4510;           // CAN message ID
        CANOL = 2;                 // message length 2 bytes
        CANOBO = (info & 0xFF);    // message data byte 0
        CANOBI = (info >> 8);      // message data byte 1
        CANOIN = 2;                // send CAN message with 29-bit ID
        swatch (secs);              // delay for specified time
        info++;                     // increment info value
    }
}

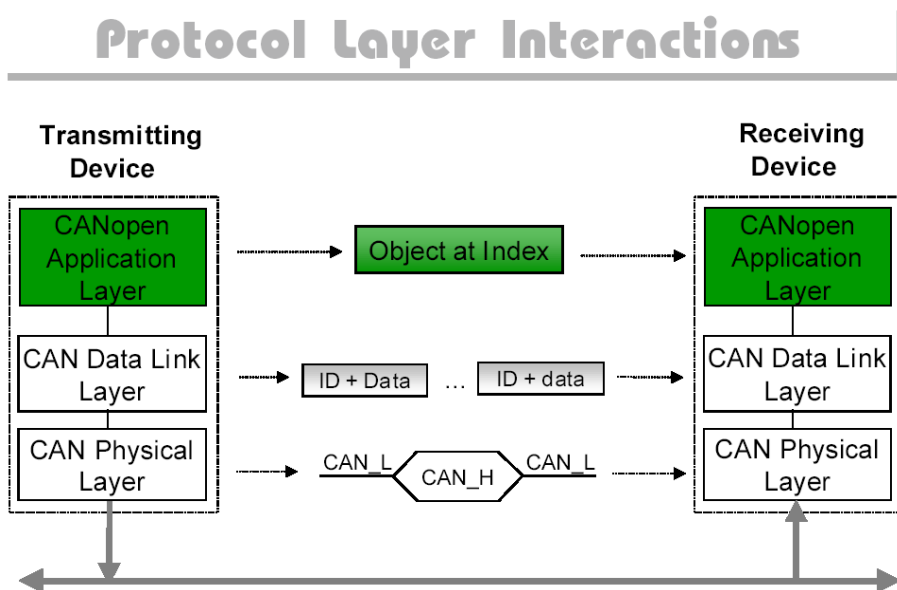
>sendCAN (0.5);                  // invoke sendCAN function
```

⁸¹ Program generovania CAN správ musí byť uložený v súbore debugger.ini odkiaľ umožňuje vytvárať jednoduché CAN správy.

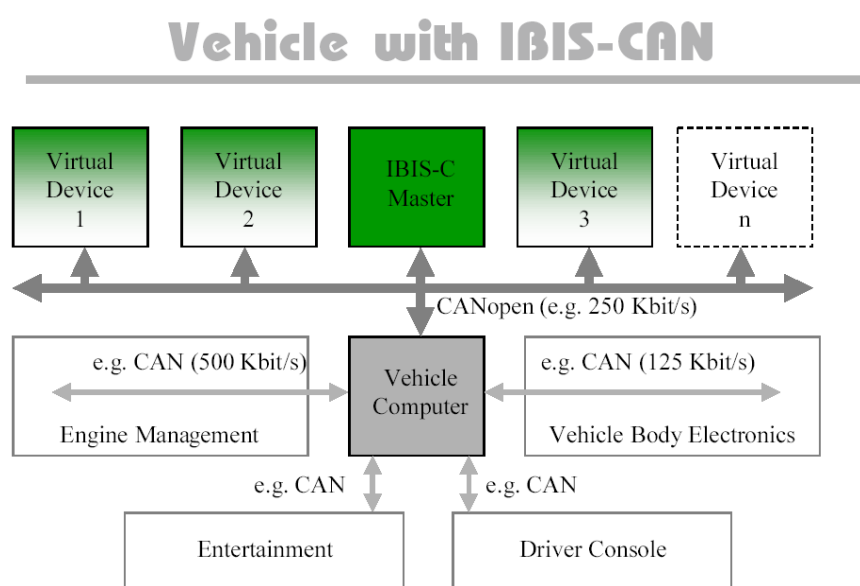
Obrázok 90, Otvorený referenčný model zbernice CAN



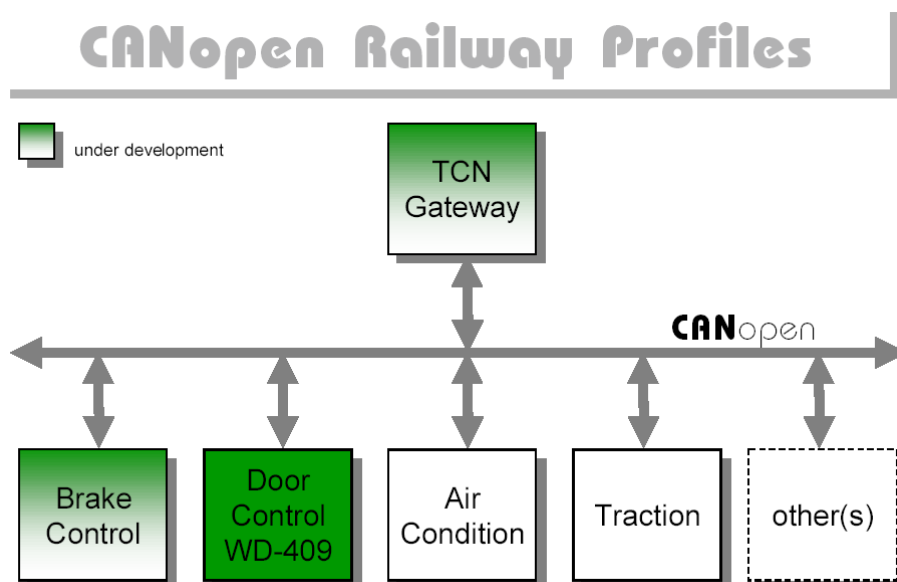
Obrázok 91, Vrstvový model zbernice CAN



Obrázok 92, Použitie zbernice CAN v automobilovom priemysle



Obrázok 93, Použitie zbernice CAN v elektrickej trakcii



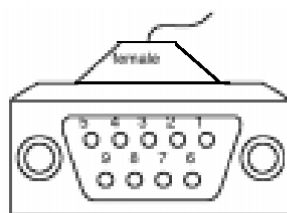
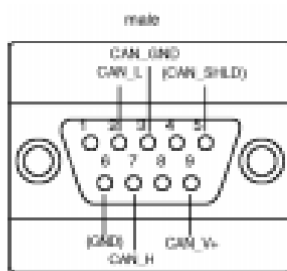
Tabuľka 9, Charakteristické vlastnosti zbernice CAN

DC Characteristics

Bus Length	Bus Cable		Termination Resistance	Max. Baudrate
	Length-Related Resistance	Bus-Line Cross-Section		
0 .. 40 m	70 mΩ/m	0.25 mm ² .. 0.34 mm ² AWG23, AWG22	124 Ω (1%)	1 Mbit/s at 40 m
40 .. 300 m	<60 mΩ/m	0.34 mm ² .. 0.6 mm ² AWG22, AWG20	127 Ω (1%)	500 Kbit/s at 100 m
300 .. 600 m	<40 mΩ/m	0.5 mm ² .. 0.6 mm ² AWG20	150 Ω to 300 Ω	100 Kbit/s at 500 m
600 m .. 1 km	<26 mΩ/m	0.75 mm ² .. 0.8 mm ² AWG 18	150 Ω to 300 Ω	50 Kbit/s at 1k m

Obrázok 94, Rozloženie vývodov zbernice CAN

CIA DS-102 Pin Assignment



9-pin D-Sub: DIN 41652

Pin	Signal	Description
1	-	Reserved
2	CAN_L	CAN_L bus line dominant low
3	CAN_GND	CAN Ground
4	-	Reserved
5	(CAN_SHLD)	Optional CAN Shield
6	GND	Optional Ground
7	CAN_H	CAN_H bus line dominant high
8	-	Reserved
9	(CAN_V+)	Optional CAN external supply

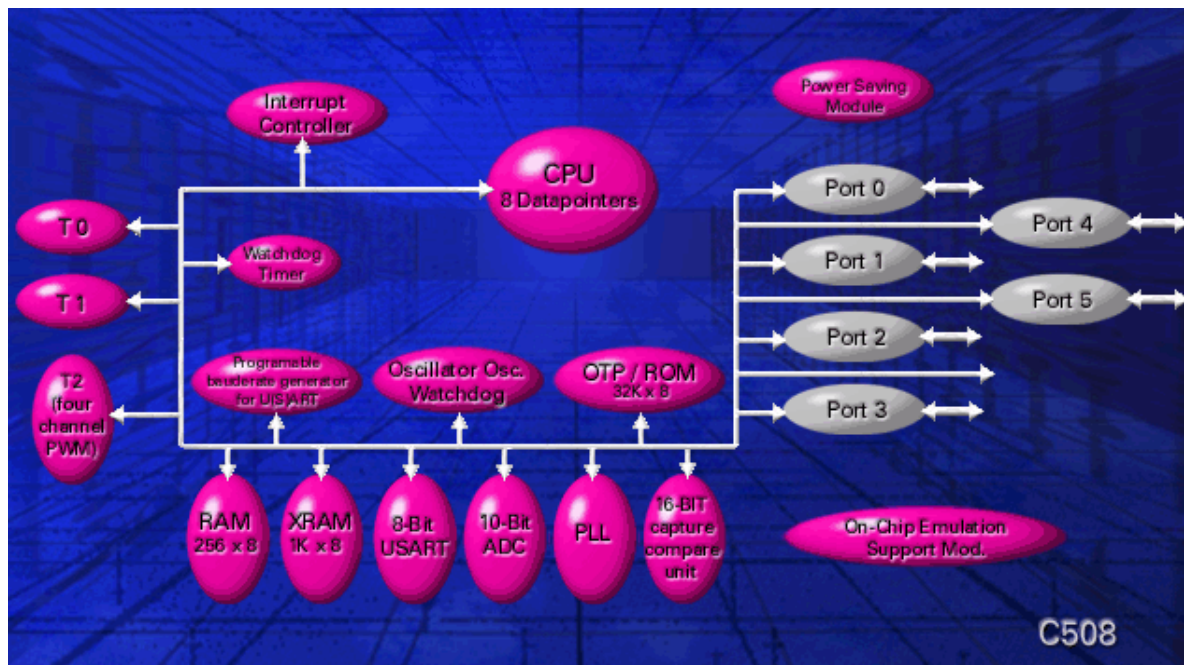
Podrobné informácie o zbernici CAN⁸² je možné získať www.can-cia.de.

⁸² <http://www.interfacebus.com/CAN-Bus-Description-Vendors-Canbus-Protocol.html>

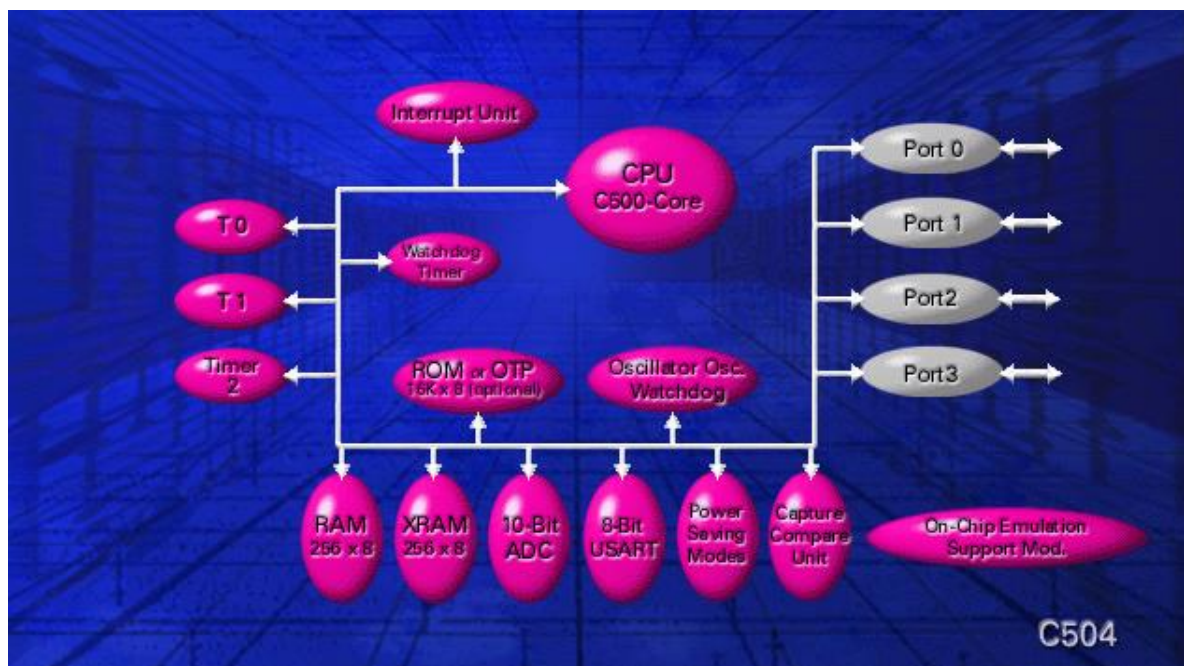
12 Moderné a perspektívne architektúry mikroprocesorov

12.1. 8 bitová architektúra

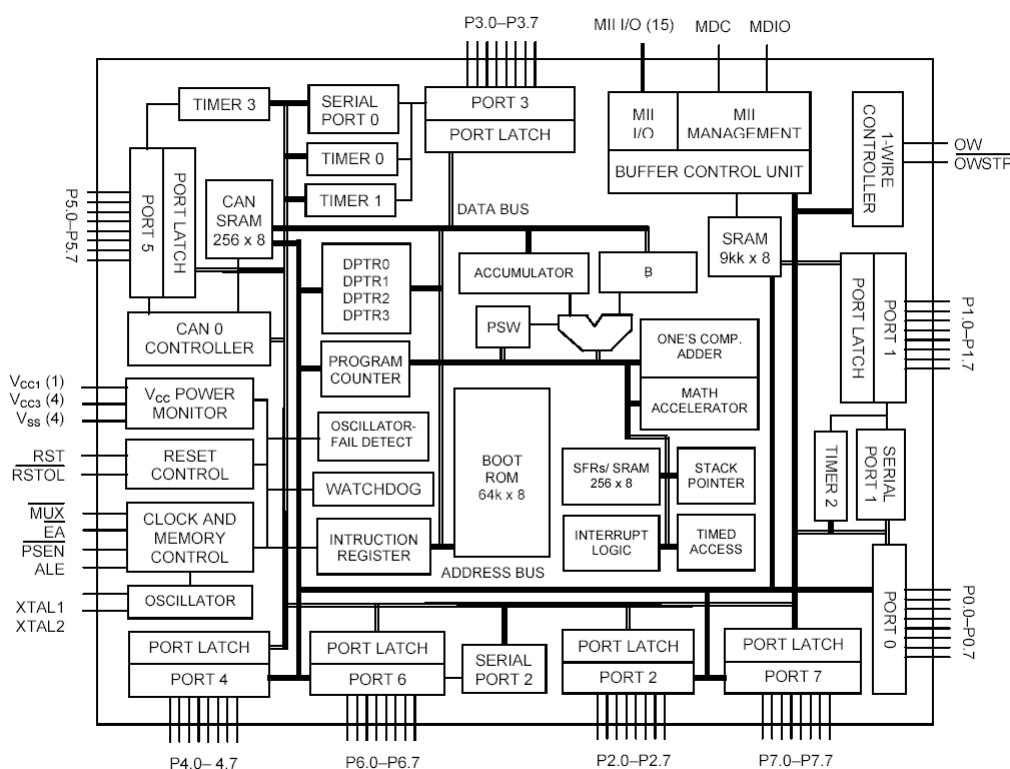
Obrázok 95, Architektúra 8051 – C508 od firmy INFINEON



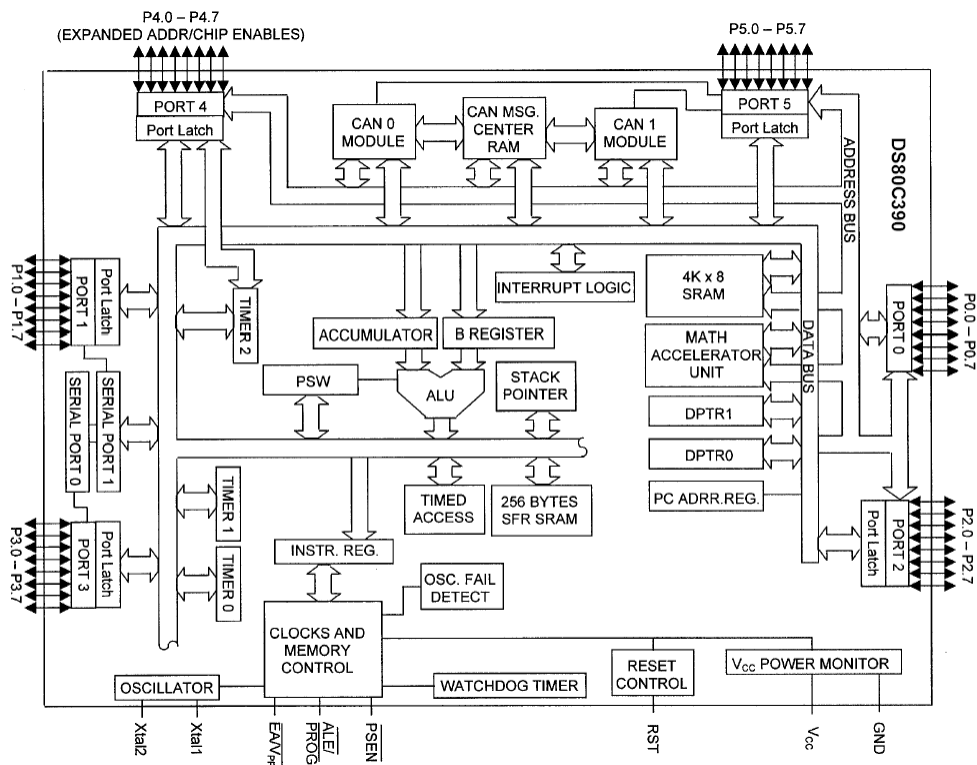
Obrázok 96, Architektúra 8051 – C504 od firmy INFINEON



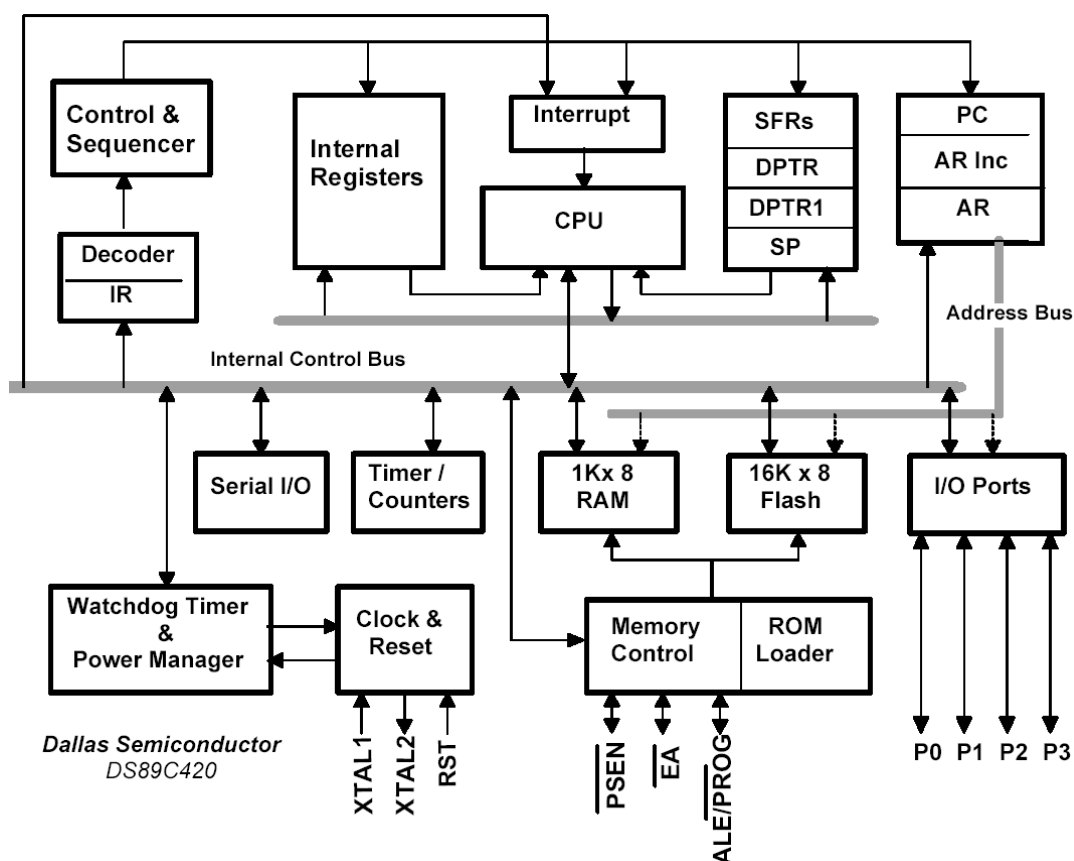
Obrázok 97, Ultra High-Speed 8051 - DS80C400 od firmy Dallas pre sieťové aplikácie



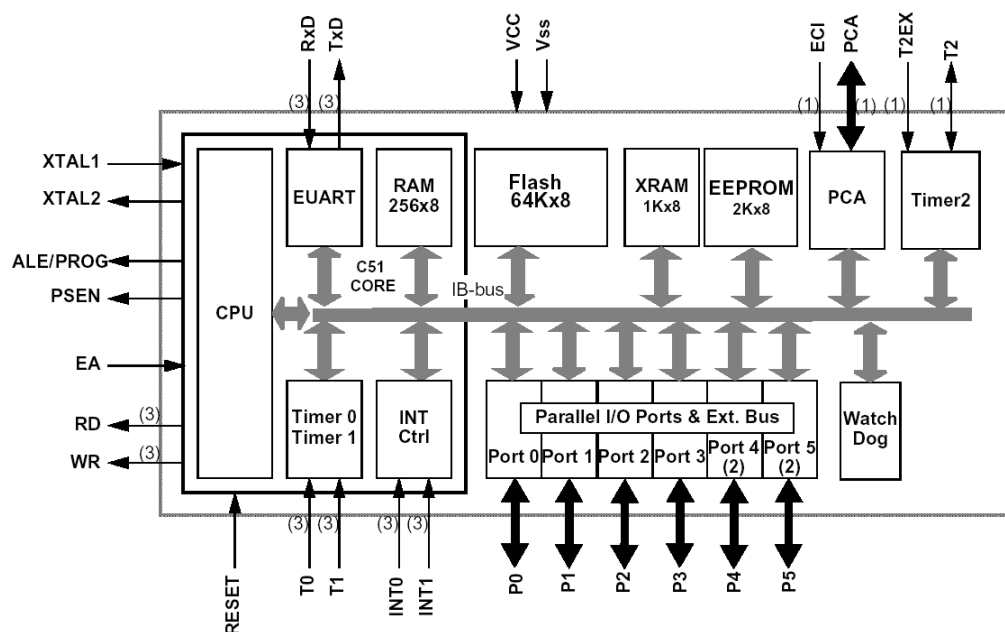
Obrázok 98, Ultra High-Speed 8051 – DS80C390 od firmy Dallas s CAN



Obrázok 99, High-Speed architektúra 8051 – DS89C420 od firmy Dallas Semiconductor

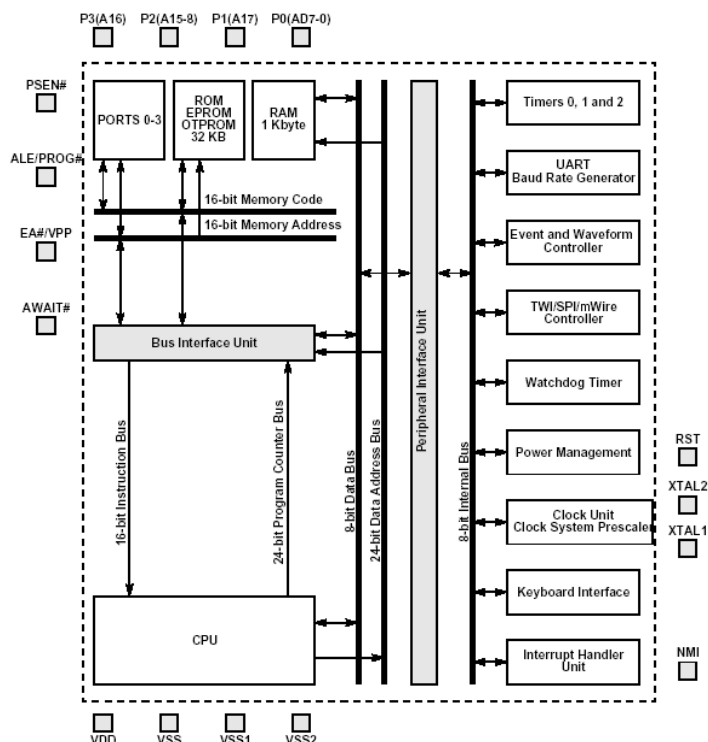


Obrázok 100, High-Speed architektúra 8051 – AT89C51RD2 od firmy ATMEL

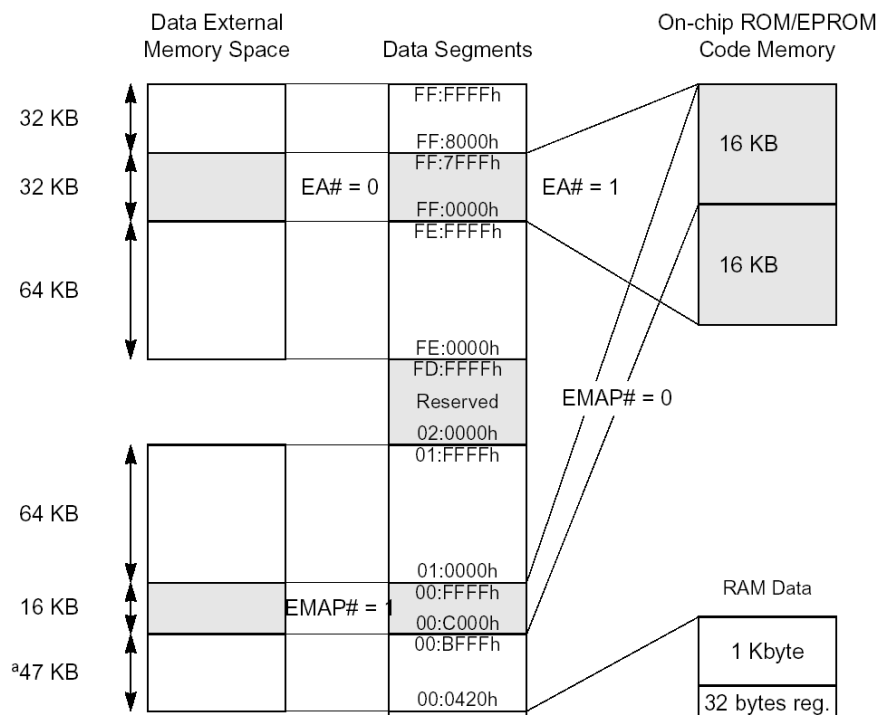


12.2. 16 bitová architektúra

Obrázok 101, High-Speed architektúra 80251 – TSC8x251G2D od firmy ATMEL

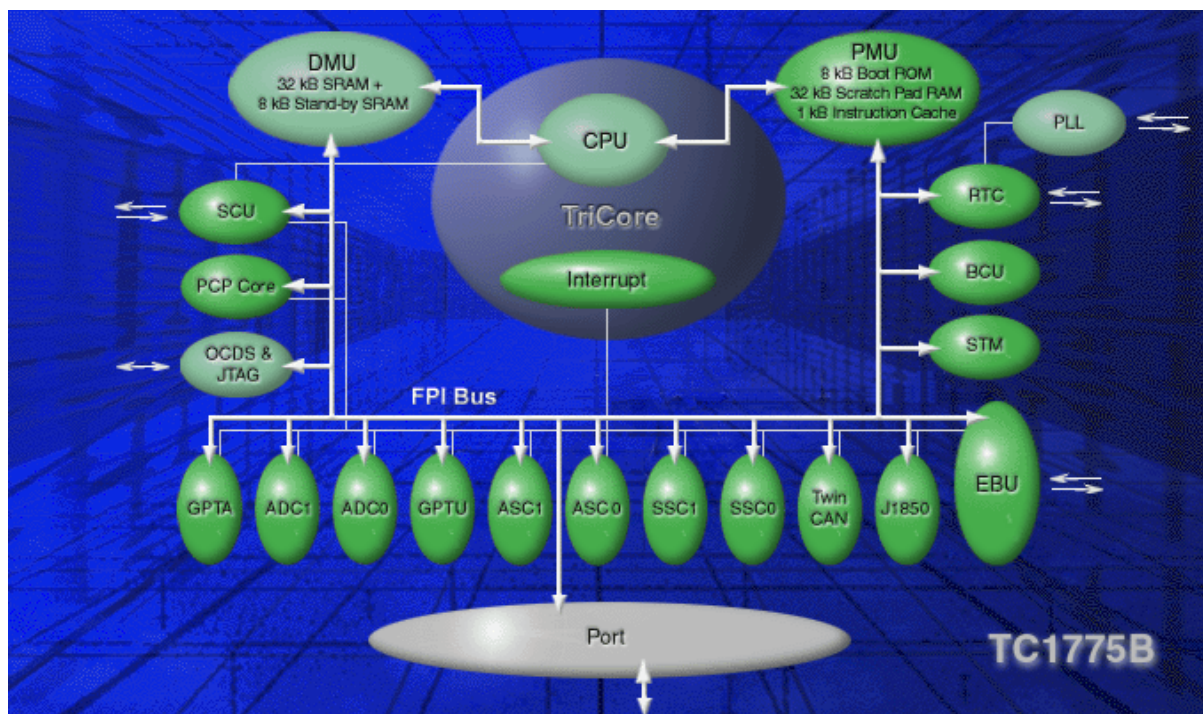


Obrázok 102, Pamäťový model architektúry 80251 – TSC8x251G2D od firmy ATMEL

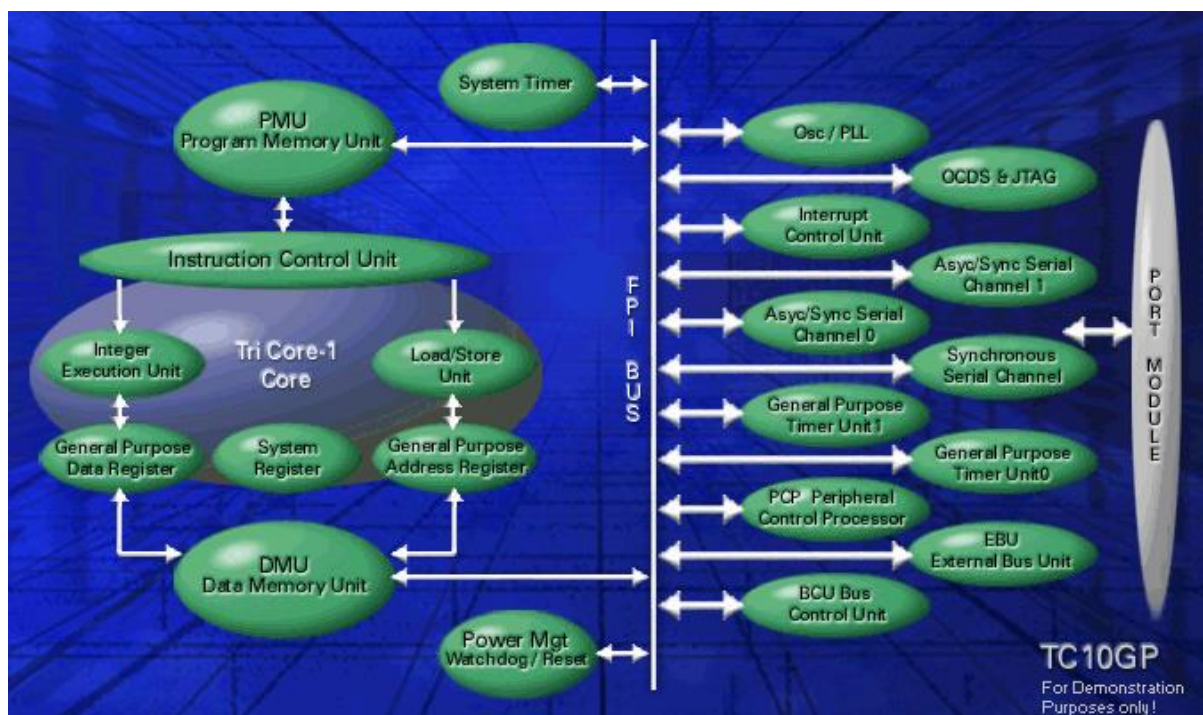


12.3. 32 bitová architektúra

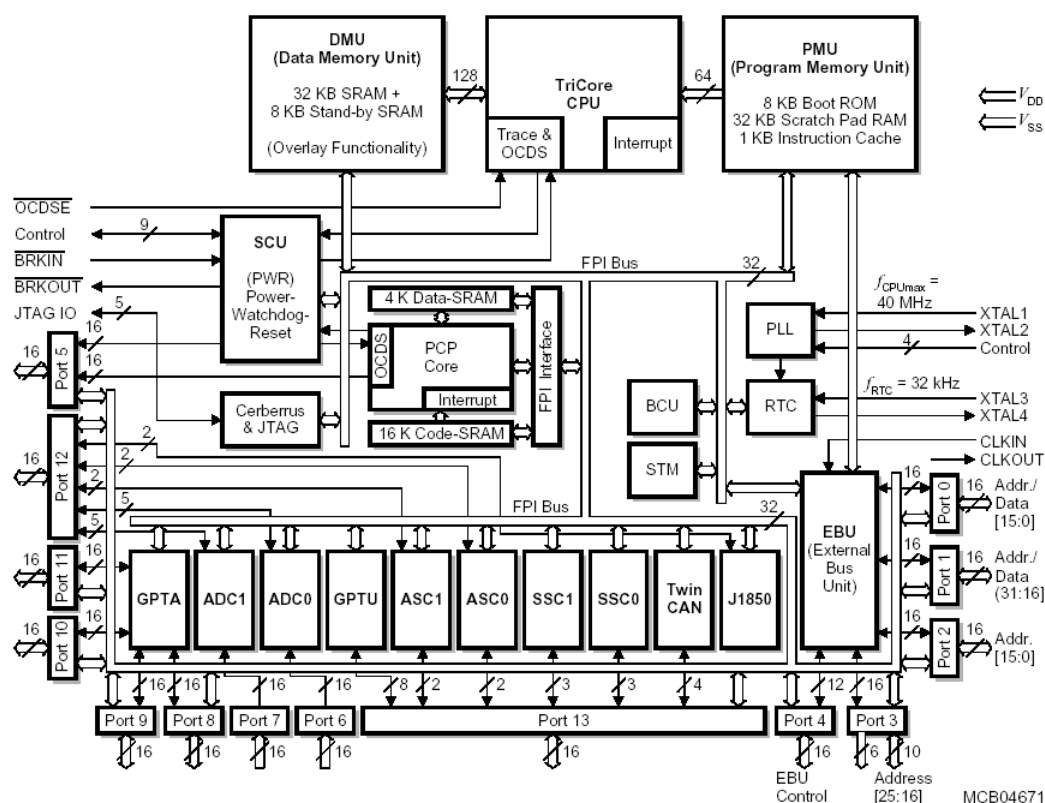
Obrázok 103, Architektúra Tri Core 32 bit. DSP TC1775B od firmy INFINEON



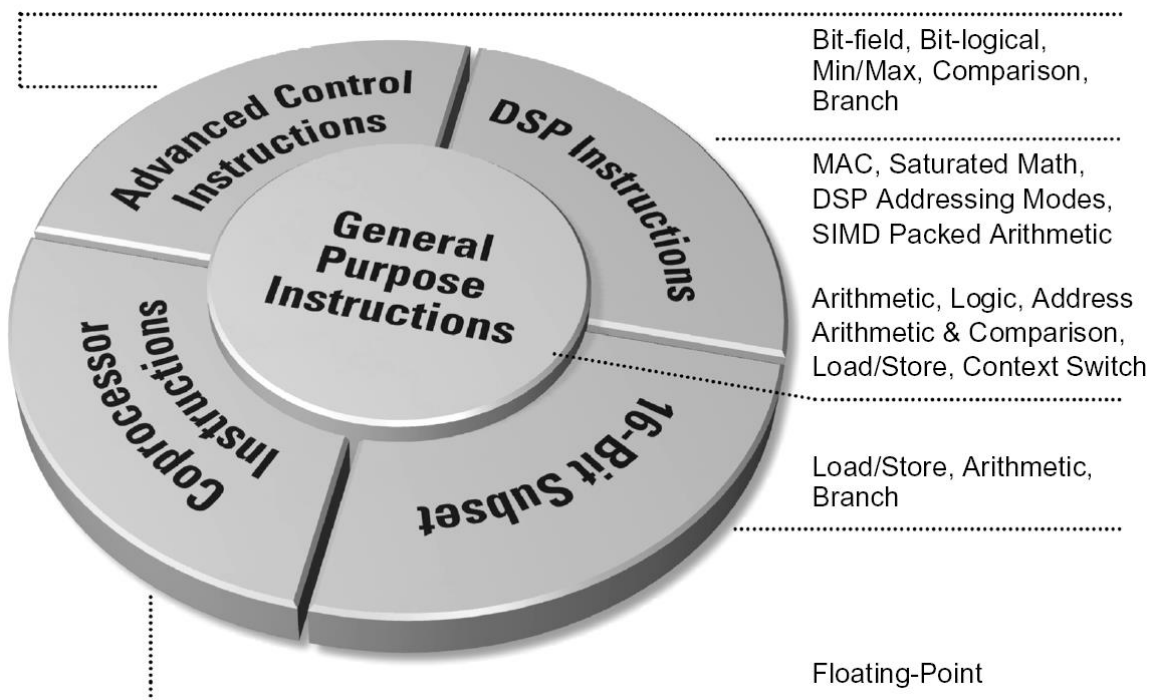
Obrázok 104, Architektúra Tri Core 32 bit. DPS TC10GP od firmy INFINEON



Obrázok 105, Architektúra 32 bit. mikroprocesora TC1775B od firmy INFINEON



Obrázok 106, Architektonické vlastnosti architektúry Tri Core od firmy INFINEON



13 Zoznam funkcií RTX51

Podrobnejší opis funkcií operačného systému RTX51 je uvedený v (Keil Elektronik GmbH, et al., 2002).

13.1. Inicializácia systému RTX51

- `os_start_system(task_number)`

13.2. Funkcie riadenia úloh

- `os_create_task(task_number)`
- `os_delete_task(task_number)`
- `os_running_task_id()`

13.3. Funkcie pre obsluhu prerušení

- `os_attach_interrupt(interrupt)`
- `os_detach_interrupt(interrupt)`
- `os_enable_isr(interrupt)`
- `os_disable_isr(interrupt)`
- `os_wait(event_selector,timeout,0)`
- `os_set_int_masks(ien0,ien1,ien2)`
- `os_reset_int_masks(ien0,ien1,ien2)`

13.4. Funkcie pre synchronizáciu úloh

- `os_send_signal(task_number)`
- `os_wait(event_selector,timeout,0)`
- `os_clear_signal(task_number)`
- `isr_send_signal(task_number)`

13.5. Funkcie pre výmenu správ medzi úlohami

- `os_send_message(mailbox,message,timeout)`
- `os_wait(event_selector,timeout,*message)`
- `isr_send_message(mailbox,message)`
- `isr_rcv_message(mailbox,*message)`

13.6. Funkcie pre zdieľanie systémových zdrojov CPU

- `os_send_token(semaphore)`
- `os_wait(event_selector,timeout,0)`

13.7. Funkcie pre správu externej pamäti

- `os_create_pool(block_size,*memory,mem_size)`
- `os_get_block(block_size)`
- `os_free_block(block_size,*block)`

13.8. Funkcie systémového časovača

- `os_set_slice(timeslice)`
- `os_wait(event_selector,timeout,0)`

13.9. Kontrolné a ladiace funkcie

- `os_check_tasks(*table)`
- `os_check_task(task_number,*table)`
- `os_check_mailboxes(*table)`
- `os_check_mailbox(mailbox,*table)`
- `os_check_semaphores(*table)`
- `os_check_semaphore(semaphore,*table)`
- `os_check_pool(block_size,*table)`

14 Programovanie aplikácií s využitím RTX51 Tiny

14.1. Problematika práce s RTX51 Tiny

V tejto kapitole nie je možné dať jednoznačný návod, ako má programátor postupovať pri tvorbe programu s využitím operačného systému RTX51, pretože to je závislé na schopnostiach programátora správne algoritmovať program. Zlému programátorovi po nevykonanom rozbere riešeného problému a nesprávneho algoritmovania nepomôže ani najnovšia, najrýchlejšia a najlepšia verzia akéhokoľvek vývojového nástroja. V tomto prípade bude platiť zlaté programátorské pravidlo, ktoré hovorí: *“Program je práve taký šikovný a dobrý ako jeho programátor”*. Hlavnou úlohou RTX51 Tiny je vykonávanie užívateľom definovaných časových intervalov jednotlivých úloh, takže sa nám môže zdať, že mikroprocesor nevykonáva inú činnosť a len čaká. *“Ponáhľ sa preto aby čakal, a čaká preto, aby sa potom ponáhľal”*.

14.2. Praktické skúsenosti s RTOS RTX51 Tiny

Po skúsenostiach s RTX51 Tiny, môžem odporučiť tým, ktorý sa rozhodnú pracovať s daným RTOS, že je vhodný práve do jednoduchých, rýchlych a nenáročných aplikácií, ktoré nevyžadujú, aby bol mikroprocesorový systém vybavený externou pamäťou dát. Z hľadiska nárokov na užívateľa vyžaduje tento systém RTOS väčšie nároky na schopnosti programátora. Užívateľ musí byť schopný napísať vlastné rutiny obsluhy prerušenia t.j. ich reakcie na výskyt prerušenia, nemôže používať zložitejšie údajové typy a štruktúry bez prítomnosti externej pamäti, absencia výmeny správ medzi úlohami (*mailbox, semaphore*) atď.

Prepnutie úloh je závislé na polohe a veľkosti zásobníka môže sa pohybovať v rozsahu 100 až 700 cyklov mikroprocesora. Odozva na prerušenie je v prípade RTX51 Tiny menšia ako 20⁸³ cyklov mikroprocesora.

Pri modifikovaných architektúrach 8051 môžu byť rozdiely, pretože používajú odlišné časovania⁸⁴ jadra mikroprocesora. S vyššie uvedenými skutočnosťami musí počítať programátor, pretože mu umožňujú znížiť reakčné časy na vzniknuté udalosti.

RTOS RTX51 Tiny je možné spustiť aj na mikroprocesoroch ktoré majú 128 Byte pamäti RAM, pričom musíme hodnotu RAMTOP uloženú v `conf_tny.a51` zmeniť na hodnotu 7Fh. Aby

⁸³ Odozva na prerušenie je závislá na nastavení parametra LONG_USR_INTR v súbore **conf_tny.a51**, kde hodnota 00h je rýchlejšia a 01h pomalšia odozva na spracovanie prerušenia.

⁸⁴ Interný delič delí frekvenciu oscilátora v pomere 1:3, 1:4, 1:6. Štandardné architektúry mikroprocesora 8051 delia frekvenciu oscilátora v pomere 1:12. UltraHighSpeed architektúry DS89C420 až DS89C450 pracujú s frekvenciou oscilátora 1:1.

RTX51 Tiny fungoval správne nesmie výsledné alokovanie aplikácií pamäti internej RAM prekročiť hodnotu cca. 75 Byte. V prípade ak nepoužívame prerušenia táto hodnota nesmie prekročiť 78 Byte RAM, ináč operačný systém RTX51 Tiny nedokáže správne prepínať úlohy navzájom a dochádza čiastočnej nefunkčnosti niektorých úloh čo je pomerne ťažké odhaliť. Jednoduchšia situácia nastáva vtedy ako alokácia pamäti dát prekročí 80 Byte, vtedy už dochádza k viditeľnej nefunkčnosti programu s RTX51 Tiny. Pri RTX51 Full je situácia odlišná, pretože už pracujeme s externou pamäťou dát a pre správnu funkčnosť si vyžaduje RTX51 cca. 900 Byte pamäti dát. Zároveň treba počítať aj cca. 100 Byte pamäti na každý task a zapuzdrené premenné.

15 Príklady programov v ASM, C51 a RTX51 Tiny

15.1. Delay v ASM ako funkcia

c:\Omega\data\Dropbox\Záloha\C51\i2c_dsw\Delay.asm

```
1  //-----
2  public      _Delay      ;Volanie funkcie s parametrom
3  //-----
4  Prog2       segment code ;Vytvaram cakaciu slucku cca. 300us pre zapis do radica HD pri 0x80
5              rseg Prog2
6  //-----
7  _Delay:     djnz r7,_Delay ;Cas do registra R7
8              ret          ;Vratim sa z rutiny podprogramu
9  //-----
10             end
11
```

Poznámka: funkcia **Delay(unsigned char)** je volaná⁸⁵ s parametrom určujúcim dĺžku oneskorenia s údajovým typom unsigned char. Príklad zápisu funkcie je **Delay(0x80)** čo pri 18,432MHz kryštáli predstavuje oneskorenie cca. 300µs.

15.2. Delay v C51 pomocou cyklu while

C:\Omega\data\Dropbox\Záloha\C51\termel\Usmernovac\Usmernovac.c

```
228 void Delay (unsigned char Dlzka) //reentrant
229 {
230     while ( (-- Dlzka ) != 0x00 );
231 }
232
```

15.3. Readkey ako funkcia

C:\Omega\data\Dropbox\Záloha\C51\Display\readkey.asm

```
1  //-----
2  public      Readkey     ;Volanie funkcie bez parametra
3  //-----
4  Prog0       segment code
5              rseg Prog0
6  //-----
7  Readkey:    jnb RI,$     ;Cakam na prijem znaku z RS232
8              mov r7,SBUF ;Ulozim ako parameter do R7
9              mov r6,#00h ;Vratim vysledok v registroch R6+R7 int
10             clr RI      ;Zmazem priznak prijmu znaku
11             ret        ;Vratim sa z rutiny podprogramu
12  //-----
13             end
14
```

Poznámka: funkcia **Readkey**⁸⁶ odovzdáva výsledok typu char volaním **Znak=Readkey()**.

⁸⁵ Funkcia **Delay()** je deklarovaná v assembler ako **public _Delay**. Podtrhnutie pred názvom funkcie vyžaduje a očakáva vstupné parametre pre funkciu (int) ktoré pri volaní musia byť umiestnené do registrov R6 a R7.

⁸⁶ **Readkey()** je obdoba funkcie **_getkey()** ktorú používa jazyk C. V samotnej podstate jazyk C odovzdáva pomocou funkcie v skutočnosti vždy len výsledok typu **int**. Údajový typ **char** použitý vo funkcii **Readkey()** je v tomto prípade vlastne pretypovaný **int** na typ **char**. Výsledok assemblerovskej funkcie je odovzdaný jazyku C pomocou registra R6 a R7. MSB je odovzdané v registri R6 a LSB v R7.

15.4. Reset v ASM

C:\Omega\data\Dropbox\Záloha\C51\Display\reset.asm

```
1 //*****
2 public      Reset
3 extrn code  (?C_STARTUP)
4 //*****
5 Prog_Reset  segment code
6             rseg Prog_Reset
7 //-----
8 Reset :     clr  ea
9             jmp  ?C_STARTUP
10            ret
11 //-----
12            end
```

15.5. Gregoriánsky kalendár v C51 ako procedúra

C:\Omega\data\Dropbox\Záloha\C51\clock full 2\Clock Full 2.c

```
241 unsigned char Gregorian(unsigned char Den,Mes,unsigned int Rok)
242 {
243     unsigned char Pocet,den,mes,nazov= Tue ;           //Tu som menil na unsigned char, zabera menej
miesta v pamati
244     unsigned int rok ;
245     for (rok=2002 ; rok <= 2099 ; rok++)
246     {
247         Datum.Tyzden =0x01 ;
248         Datum.Poradie =0x01 ;
249         for (mes=Jan ; mes <= Dec ; mes++)
250         {
251             // os_wait(K_TMO,1,NULL);
252             Pocet=Pocet_Dni [mes-1];
253             if (mes == Feb) {if (rok %4==0) Pocet=29; else Pocet=28;}
254             for (den=1; den <= Pocet ; den++)
255             {
256                 if ((Den == den) && (Mes == mes) && (Rok == rok))
257                 {
258                     if (((Rok %4) != 0) && (Mes >= 3)) Datum.Poradie++; //Korekcia mien na prestupnost rokov
259                     return (nazov);
260                 }
261                 if (nazov++>= Sun) {nazov=Mon; Datum.Tyzden++;}
262                 if (Datum.Tyzden >53) Datum.Tyzden=0x01;           //Bolo tu toto asi zle: Datum.Tyzden>=53
263                 Datum.Poradie++;
264             }
265         }
266     }
267     return (nazov);
268 }
269
270 void Vypocet_Datumu (void)
271 {
272     Datum.Nazov=Gregorian (Datum.Den,Datum.Mes,Datum.Rok);
273     // Datum.Nazov=Gregor(Datum.Den,Datum.Mes,(Datum.Rok-1900));
274 }
```

Poznámka: Vyššie uvedené procedúra pre výpočet poradového čísla dňa je veľmi podobná ako vo funkcii programu napísaného v assembler-i uvedeného v kapitole 15.6.

15.6. Gregoriánsky kalendár v ASM ako funkcia

C:\Omega\data\Dropbox\Záloha\C51\clock full 2\KALENDAR.ASM

```
1 public _Gregor
2 ;-----
3 Vars segment data
4 rseg Vars
5 ;-----
6 Den: ds 1
7 Mes: ds 1
8 Rok: ds 1
9 Poc_Dni: ds 1
10 Akt_Den: ds 1
11 Nas_Den: ds 1
12 Nas_Mes: ds 1
13 Nas_Rok: ds 1
14 ;-----
15 Tabs segment code
16 rseg Tabs
17 ;-----
18 Tab1: db 00h,31d,29d,31d,30d,31d,30d,31d,30d,31d,30d,31d ;Priestupny rok ...
19 Tab2: db 00h,31d,28d,31d,30d,31d,30d,31d,30d,31d,30d,31d ;Normalny rok ...
20 ;-----
21 Progs0 segment code
22 rseg Progs0
23 ;-----
24 _Gregor: mov Nas_Den,r7 ;r7+r6
25 mov Nas_Mes,r5 ;r5+r4
26 mov Nas_Rok,r3 ;r3+r2 24.2.2002, udava sa v tvare Rok-1900 !!!
27 mov Rok,#01h ;1901, rok 101 je 2001
28 mov Mes,#01h ;Januar
29 mov Den,#01h ;1.1.1901
30 mov Akt_Den,#02h ;1.1.1901 bol Utorok
31 ?PR?Greg: mov a,Rok
32 mov b,#04h
33 div ab
34 mov a,b
35 cjne a,#00h,PreNo
36 PreYes: mov dptr,#Tab1 ;Pocet dni v mesiaci pre rok
37 jmp PreEnd
38 PreNo: mov dptr,#Tab2 ;Pocet dni v mesiaci pre rok
39 ;-----
40 PreEnd: mov a,Mes
41 movc a,@a+dptr ;Z TabX vyberiem spravny pocet dni v mesiaci
42 mov Poc_Dni,a
43 mov a,Den
44 mov r0,#Akt_Den ;Nepriamo adresujem premennu Akt_Den
45 jmp Cyklus
46 ?PR?XXX: inc @r0
47 inc Den
48 mov a,Den
49 cjne @r0,#08h,Cyklus
50 mov @r0,#01h
51 ;-----
52 Cyklus: push acc
53 mov a,Den
54 cjne a,Nas_Den,?PR?L111 ;Tu mozem zastavit na aktualnom datume XX.XX.?PR?XXXX !!!
55 mov a,Mes
56 cjne a,Nas_Mes,?PR?L111
57 mov a,Rok
58 cjne a,Nas_Rok,?PR?L111
59 pop acc ;Ak sa dostanem tu, nasiel som akt. datum
60 mov r7,Akt_Den
61 ret ;Ukoncujem funkciu
62 ;-----
63 ?PR?L111: pop acc
64 cjne a,Poc_Dni,?PR?XXX ;Tu mozem zistit den pripadajuci na 31.1.1901 !!!
65 inc Akt_Den ;Prechadzam do dalsieho mesiaca, tak aktualizujem aj Akt_Den
66 mov r0,#Akt_Den ;Pracujem s nepriamou adresaciou, pretoze je to teraz vyhodne
67 cjne @r0,#08h,?PR?YYY ;Aby nahodou nepridalo osmy den ....
68 mov Akt_Den,#01h ;Ak je osmy den, potom nastav na 1
69 ?PR?YYY: mov Den,#01h
70 mov r0,#Mes
71 inc @r0 ;Zvysim o jeden mesiac
72 cjne @r0,#13d,?PR?Greg ;Uz som presiel 12 mesiac, musim pretocit na 1 mesiac
73 mov @r0,#01h ;Nastavim 1 mesiac
74 inc Rok
75 jmp ?PR?Greg
76 ?PR?Cycl: call ?PR?Greg
77 inc Rok ;Ak som doteraz nenasiel idem na dalsi rok
78 jmp ?PR?Cycl
79 end
```

Poznámka: funkcia **unsigned char Gregor(unsigned char, unsigned char, unsigned char)** vracia číslo v rozsahu 1 až 7 predstavujúce deň v týždni. Od pondelka do nedele.

15.7. Bitový invertor portu

V praxi je niekedy potrebné medzi sebou zameniť jednotlivé významové bity portu medzi sebou. Na internetových stránkach⁸⁷ diskusných skupín je možné nájsť množstvo spôsobov ako zameniť bity medzi sebou. Niektoré využívajú metódy výpočtu⁸⁸ a ďalšie využívajú tabuľkové spôsoby spracovania údajov. Uvedený program využíva metódu výberu a náhradu dát z tabuľky, čím dosahujeme vysokú efektivitu a rýchlosť⁸⁹ oproti výpočtovým metódam. Pri vstupných hodnotách „abcdefgh” dostaneme na výstupe „hgfedcba“.

Obrázok 107, Bitový invertor portu

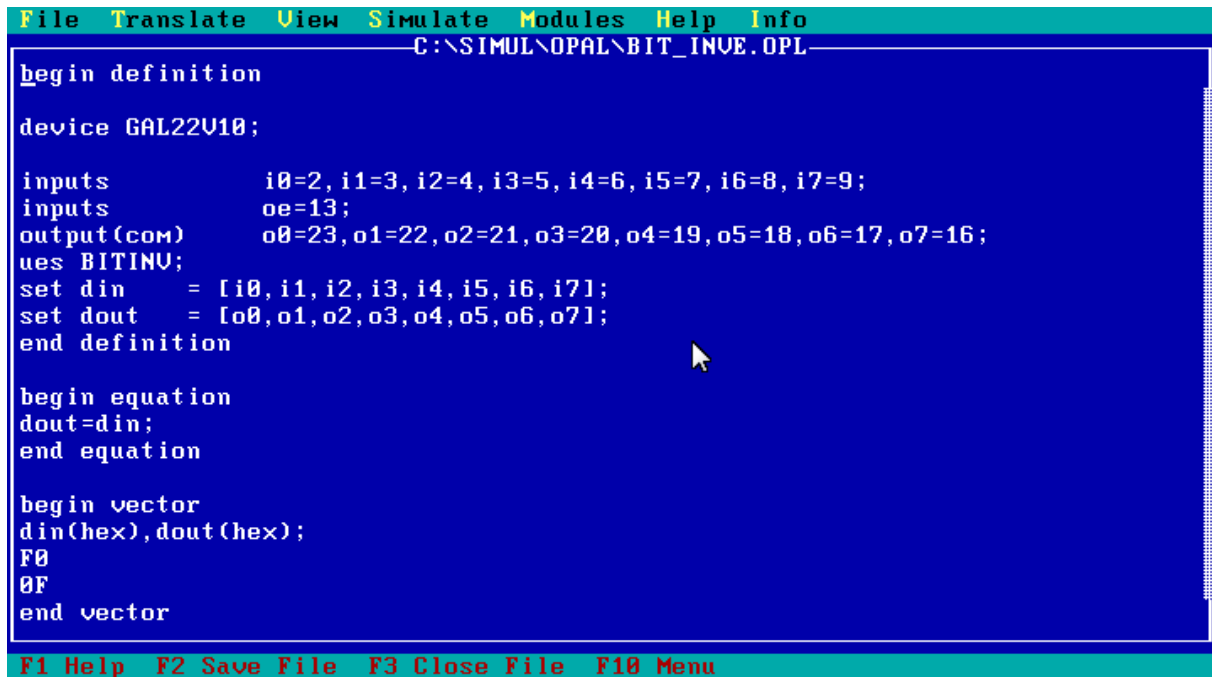
```
1  #include <reg52.h>
2  #include <stdio.h>
3  #include <stdlib.h>
4
5
6  unsigned char code Table[]=
7  {
8      0x00,0x80,0x40,0xC0,0x20,0xA0,0x60,0xE0,
9      0x10,0x90,0x50,0xD0,0x30,0xB0,0x70,0xF0,
10     0x08,0x88,0x48,0xC8,0x28,0xA8,0x68,0xE8,
11     0x18,0x98,0x58,0xD8,0x38,0xB8,0x78,0xF8,
12     0x04,0x84,0x44,0xC4,0x24,0xA4,0x64,0xE4,
13     0x14,0x94,0x54,0xD4,0x34,0xB4,0x74,0xF4,
14     0x0C,0x8C,0x4C,0xCC,0x2C,0xAC,0x6C,0xEC,
15     0x1C,0x9C,0x5C,0xDC,0x3C,0xBC,0x7C,0xFC,
16     0x02,0x82,0x42,0xC2,0x22,0xA2,0x62,0xE2,
17     0x12,0x92,0x52,0xD2,0x32,0xB2,0x72,0xF2,
18     0x0A,0x8A,0x4A,0xCA,0x2A,0xAA,0x6A,0xEA,
19     0x1A,0x9A,0x5A,0xDA,0x3A,0xBA,0x7A,0xFA,
20     0x06,0x86,0x46,0xC6,0x26,0xA6,0x66,0xE6,
21     0x16,0x96,0x56,0xD6,0x36,0xB6,0x76,0xF6,
22     0x0E,0x8E,0x4E,0xCE,0x2E,0xAE,0x6E,0xEE,
23     0x1E,0x9E,0x5E,0xDE,0x3E,0xBE,0x7E,0xFE,
24     0x01,0x81,0x41,0xC1,0x21,0xA1,0x61,0xE1,
25     0x11,0x91,0x51,0xD1,0x31,0xB1,0x71,0xF1,
26     0x09,0x89,0x49,0xC9,0x29,0xA9,0x69,0xE9,
27     0x19,0x99,0x59,0xD9,0x39,0xB9,0x79,0xF9,
28     0x05,0x85,0x45,0xC5,0x25,0xA5,0x65,0xE5,
29     0x15,0x95,0x55,0xD5,0x35,0xB5,0x75,0xF5,
30     0x0D,0x8D,0x4D,0xCD,0x2D,0xAD,0x6D,0xED,
31     0x1D,0x9D,0x5D,0xDD,0x3D,0xBD,0x7D,0xFD,
32     0x03,0x83,0x43,0xC3,0x23,0xA3,0x63,0xE3,
33     0x13,0x93,0x53,0xD3,0x33,0xB3,0x73,0xF3,
34     0x0B,0x8B,0x4B,0xCB,0x2B,0xAB,0x6B,0xEB,
35     0x1B,0x9B,0x5B,0xDB,0x3B,0xBB,0x7B,0xFB,
36     0x07,0x87,0x47,0xC7,0x27,0xA7,0x67,0xE7,
37     0x17,0x97,0x57,0xD7,0x37,0xB7,0x77,0xF7,
38     0x0F,0x8F,0x4F,0xCF,0x2F,0xAF,0x6F,0xEF,
39     0x1F,0x9F,0x5F,0xDF,0x3F,0xBF,0x7F,0xFF,
40 };
41
42 unsigned char i;
43
44 void main(void)
45 {
46     while(1)
47     {
48         for(i=0x00; i<=255; i++)
49         {
50             P1=Table[i];
51         }
52     }
53 }
```

⁸⁷ <http://www.keil.com/forum/16883/>

⁸⁸ <http://8052.com/forum/read/90817>

⁸⁹ Čas spracovania algoritmu výberu z tabuľky trvá približne 11 cyklov mikroprocesora, čo pri 12MHz kryštáli predstavuje typicky približne 11μs čo je oproti iným technikám minimálne dvojnásobné urýchlenie. Ak by sme požadovali rýchlosti pod 10ns je potrebné využiť programovateľné hradlové pole napr. GAL22V10 ktoré je schopné tento bitový invertor vytvoriť pomocou programovateľných logických hradíel.

Obrázok 108, Bitový invertor s EPLD



```
File Translate View Simulate Modules Help Info
C:\SIMUL\OPAL\BIT_INVE.OPL
begin definition
device GAL22V10;

inputs      i0=2, i1=3, i2=4, i3=5, i4=6, i5=7, i6=8, i7=9;
inputs      oe=13;
output(com) o0=23, o1=22, o2=21, o3=20, o4=19, o5=18, o6=17, o7=16;
ues BITINV;
set din     = [i0, i1, i2, i3, i4, i5, i6, i7];
set dout    = [o0, o1, o2, o3, o4, o5, o6, o7];
end definition

begin equation
dout=din;
end equation

begin vector
din(hex), dout(hex);
F0
0F
end vector

F1 Help  F2 Save File  F3 Close File  F10 Menu
```

Programovateľné logické polia tvoria v elektronike významnú skupinu súčiastok, ktorých funkcia sa dá programovo meniť. Väčšinou plnia funkciu kombinačných, alebo sekvenčných logických obvodov. Modernejšie obvody MAPL128⁹⁰ už dokážu vytvoriť aj zložitejšie elektronické konštrukcie napr. arbiter ktorý riadi asynchrónne prístup dvoch procesorov do jednej pamäti RAM.

⁹⁰ Texas Instruments, www.ti.com

15.8. Ovládač sériového rozhrania RS232 v C51 bez riadenia prerušením

```
1  /*-----*/
2  HELLO.C
3
4  Copyright 1995-2005 Keil Software, Inc.
5  /*-----*/
6
7  #include <REG52.H>          /* special function register declarations */
8                             /* for the intended 8051 derivative */
9
10 #include <stdio.h>          /* prototype declarations for I/O functions */
11
12
13 #ifdef MONITOR51            /* Debugging with Monitor-51 needs */
14 char code reserve [3] _at_ 0x23; /* space for serial interrupt if */
15 #endif                      /* Stop Execution with Serial Intr. */
16                             /* is enabled */
17
18
19 /*-----*/
20 The main C function. Program execution starts
21 here after stack initialization.
22 /*-----*/
23 void main (void) {
24
25 /*-----*/
26 Setup the serial port for 1200 baud at 16MHz.
27 /*-----*/
28 #ifndef MONITOR51
29     SCON = 0x50;          /* SCON: mode 1, 8-bit UART, enable rcvr */
30     TMOD |= 0x20;         /* TMOD: timer 1, mode 2, 8-bit reload */
31     TH1 = 221;           /* TH1: reload value for 1200 baud @ 16MHz */
32     TR1 = 1;             /* TR1: timer 1 run */
33     TI = 1;              /* TI: set TI to send first char of UART */
34 #endif
35
36 /*-----*/
37 Note that an embedded program never exits (because
38 there is no operating system to return to). It
39 must loop and execute forever.
40 /*-----*/
41 while (1)
42 {
43     P1 ^= 0x01;          /* Toggle P1.0 each time we print */
44     printf ("Hello World\n"); /* Print "Hello World" */
45 }
46 }
```

Popis programu: Vyššie uvedený program využíva štandardnú knižnicu na výpis textového reťazca pomocou funkcie `printf("Hello World\n")`⁹¹ reťazec "Hello world" s prenosovou rýchlosťou 19200 b.s⁻¹. Pri použití RTX51 Tiny nie je potrebné vykonať žiadne zmeny v programe. Jediné obmedzenie⁹² použitie funkcie `printf()` v RTX51 Tiny je podstatné v tom, že nereentrantnú⁹³ funkciu je možné použiť len v jednom task-u.

⁹¹ Pri použití funkcie **printf()** musí byť správne inicializované sériové rozhranie RS232 a na záver inicializačnej sekvencie nastavený bit **TI=1**, v opačnom prípade funkcia nebude vôbec fungovať. Bit TI=1 musíme nastaviť len vtedy ak nepoužívame pri prenose dát prerušenie od sériového kanála.

⁹² Obmedzenie je možné odstrániť povolením overlay-u a zabezpečiť tak viacnásobné volanie funkcie viacerými task-ami s RTX51 Full. V menu **BL51 Misc** povoliť overlay funkcie príkazom: `ol(?PR?PRINTF?PRINTF ! *)`, čo umožní viacnásobný prístup z iných task-ov k funkcii `printf()`. Pritom je treba zabezpečiť, aby nedošlo k súčasnému prístupu úloh k funkcii `printf` !!! Pre názornosť uvádzam zápis tzv. viacnásobného použitia overlay-u vo viacerých taskoch programu s použitím operačného systému RTX51 Full: `ol(?PR?_PUT_STRING?CLOCK_FULL_2 ! *,?PR?_PUT_INS?CLOCK_FULL_2 ! *,?PR?PRINTF?PRINTF ! *)`.

⁹³ Funkcie ktoré môžu byť volané reentrant-ne t.j. súčasne z viacerých zdrojov využívajú definovaný zásobník pre odkladanie environment-u prostredia funkcie. Bez deklarácie reentrant pri volaní funkcie dôjde k nevyhnutnému poškodeniu vlastných premenných funkcie a tým k nesprávnej funkcii.

15.9. Ovládač 232.NET pre multiprocessorovú komunikáciu

C:\Omega\Data\Dropbox\Záloha\C51\Multiprocessorová komunikácia s 8051\Multiproce

```
1  /*****
2  * 232.NET multiprocessorová komunikácia *
3  *****/
4
5  #include <reg552.h>
6  #include <stdio.h>
7  #include <stdlib.h>
8  #include <string.h>
9  #include <rtx51.h>
10 #include <dsb.h>
11
12 #define Token      '^' //Malo by to byť 0x5E='^'
13 #define MyAddrNet  0x00 //Definujem adresu siete 8051
14 #define MyAddr51   0xAB //Definujem moju adresu, dostanem ju z .ini debuggera-a
15 #define MyBroadcast 0xFF //Definujem Broadcast-s address-u
16 #define WaitForever 0xFF //Čakam donekonečna ...
17 #define MBXSend    0x00 //Číslo mailbox-u kde sa odosiela data
18 #define MBXRecv    0x01 //Číslo mailbox-u kde sa prijímajú data
19
20 #define SYSTEM     0x00
21 #define SERIAL     0x01
22 #define COM        0x02
23
24 extern void Reset51(void);
25
26 #define SizeOfPacketData 32
27
28 typedef xdata struct //Musi byť deklarovaná a používaná
29 ako je to uvedené v tomto tvare, inak správne nefunguje !!!
30 {
31     unsigned char Preamble,Dst,Src;
32     unsigned int Size;
33     unsigned char Data[SizeOfPacketData];
34     unsigned char Crc;
35 }
36 Packet51;
37
38 #define NumMemBlocks 5 //Definujem [n] pamäťových blokov
39 ...
40 #define SizeMemBlock (sizeof(Packet51))
41 #define SizeMemPool (NumMemBlocks*(2+SizeMemBlock)) // (Header+sizeof(Packet51)*n)
42
43 xdata volatile Packet51 Packet; //Definoval som prázdny packet,
44 naplním ho v inom task-u, štruktúra nesmie byť optimalizovaná kompilátorom ...
45 xdata unsigned char MemoryPool[SizeMemPool] _at_ 0x0000; //MemoryPool je niekedy lepšie
46 definovať na pevnej adrese ale uVision robí problémy pri simulácii ...
47 xdata Packet51 *TransmitMessage,*ReceivedRS232; //Ukazovateľ na prijaté znaky z
48 mailbox-u a na prijaté znaky z RS232 a obsahujem prijatý packet z RS232 ...
49 xdata unsigned int *MessageToSend,*MessageToRecv; //Rozne nadefinované buffer-e
50 vysielaného a prijímaného packet-u
51
52 enum state {OFF=0, ON=1};
53 enum lists {preamblepkt=0, dstpkt=1, srcpkt=2, sizeofpkthi=3, sizeofpktlo=4, headerpkt=5,
54 crcpkt=(sizeof(Packet51)-1), endpkt=sizeof(Packet51), sizedatapkt=SizeOfPacketData,
55 mempoolpkt=sizeof(MemoryPool)};
56
57 void Com(void) _task_ COM _priority_ 0
58 {
59     signed char RtxCompletion;
60     while(1)
61     {
62         switch(RtxCompletion=os_wait(K_TMO+K_SIG+K_INT,WaitForever,NULL))
63         {
64             case TMO_EVENT :
65             case SIG_EVENT : RtxCompletion=0x00;break; //Prisli mi data z RS232 ...
66             case INT_EVENT : break;
67         }
68     }
69 }
70
71 unsigned char Check_CRC(unsigned char *Block) reentrant //Volam rutinu z viacerých
72 taskov ...
73 {
74     unsigned int i;
75     unsigned char tmp;
76 }
```

Page 1

C:\Omega\Data\Dropbox\Záloha\C51\Multiprocessorová komunikácia s 8051\Multiproce

```
67 tmp=0x00;
68 for(i=0x00; i<(sizeof(Packet51)-1); i++) tmp+=Block[i]; //Scitam všetky položky packetu
69 tmp=~tmp; tmp+=1;
70 return(tmp);
71 }
72
73 bit recv=OFF, send=OFF;
74
75 void Serial(void) _task_ SERIAL _priority_ 0
76 {
77     signed char RtxCompletion;
78     idata unsigned int iin,iout;
79     data unsigned char Znak;
80     TMOD|=0x20;
81     PCON|=0x80;
82     SOCON|=0x50;
83     TH1=0xFB; //Nastavim prenosovu
84     rychlost 19200bps
85     TR1=1; //TI=1; //Je tu potrebné len pre
86     printf(), putchar() !!!
87     if((MessageToSend=os_get_block(SizeMemBlock))!=NULL); //Blok pamati rezervujem
88     pre task ktory je pouzity v SERIAL
89     if((MessageToRecv=os_get_block(SizeMemBlock))!=NULL) ReceivedRS232=(unsigned
90     int)MessageToRecv; //Adresa prijimacieho buffera v pool-e sa inicializuje ...
91     RtxCompletion=os_attach_interrupt(4); //Simultanny prijem a
92     vysielanie je riadene prerusenim od serioveho kanala ...
93     while(1)
94     {
95         switch(RtxCompletion=os_wait(K_TMO+K_SIG+K_INT+K_MBX+MBXSend,WaitForever,(unsigned int
96         xdata*)&TransmitMessage)) //Prijem packetu z mailbox-u, prijimam len adresu na strukturu ...
97         {
98             case TMO_EVENT : iin=0x0000; //Ak nepride
99             prerusenie od prijmu dat nulujem pocitadlo aby som sa nezacyklil ...
100             //memset(ReceivedRS232,NULL,sizeof(Packet51));
101             //Vynulujem prijimaci buffer ReceiveRS232 ...
102             break;
103             case MSG_EVENT : //os_send_signal(SERIAL); break;
104             //Nepotrebujem pocitat CRC prijateho packetu a predsa to posielam v ramci 8051, aby som poslal
105             signal sam sebe na spracovanie ... !!!
106             if(Check_CRC((unsigned char
107             xdata*)&TransmitMessage)==(TransmitMessage->Crc)) os_send_signal(SERIAL); break; //Vypocitam
108             CRC prijateho packetu a musia byt rovnake aby som poslal signal sam sebe na spracovanie ... !!!
109             break;
110             case SIG_EVENT : send=ON; iout=preamblepkt;
111             SOBUF=(TransmitMessage->Preamble);
112             os_wait(K_TMO,0,NULL); //Aby stihli
113             všetky systémy zaregistrovať prítomnosť Tokenu '^' na sieti, majú niekoľko ms...
114             break;
115             case INT_EVENT : if(TI&RI) recv=send=ON; //Ak som tu,
116             tak mám prijem a vysielanie súčasne čo potvrdzuje súčasne prijímanie a vysielanie súčasne ...
117             if(RI)
118             {
119                 RI=0; Znak=SOBUF; recv=ON; //Prítomnosť
120                 tokenu signalizuje data na zbernici
121                 if(Znak==Token)
122                 {
123                     iin=0x0000;
124                     memset(ReceivedRS232,NULL,sizeof(Packet51)); //Vynulujem
125                     prijimaci buffer ReceiveRS232, pretože obsahuje stare hodnoty ...
126                     switch(iin++)
127                     {
128                         case preamblepkt : ReceivedRS232->Preamble=Znak; break; //Ak je to
129                         token tak sa nastavím na začiatok dat ...
130                         case dstpkt : ReceivedRS232->Dst=Znak; break;
131                         case srcpkt : ReceivedRS232->Src=Znak; break;
132                         case sizeofpktthi : ReceivedRS232->Size=256*Znak; break;
133                         case sizeofpktlo : ReceivedRS232->Size+=Znak; break;
134                         case crcpkt : ReceivedRS232->Crc=Znak;
135                         if(Check_CRC((unsigned char
136                         xdata*)&ReceivedRS232)==(ReceivedRS232->Crc))
137                         {
138                             recv=OFF;
139                         }
140                     }
141                 }
142             }
143             //RtxCompletion=os_free_block(SizeMemBlock,MessageToRecv); //Zrusím prijimaci buffer ...
144         }
145     }
146 }
```

Page 2

C:\Omega\Data\Dropbox\Záloha\C51\Multiprocesorová komunikácia s 8051\Multiproce

```
123      switch(ReceivedRS232->Dst)           //Komu je
124      packet urceny, vykonam akciu ???
125      {
126          case MyAddrNet : break;
127          case MyAddr51  :
128          case MyBroadcast :
129              RtxCompletion=os_send_signal(COM); //V strukture ReceivedRS232 je prijaty
130              packet ...
131              break;
132          default :
133              memset(ReceivedRS232,NULL,sizeof(Packet51)); break; //Nie su to data pre mna, mazem prijimaci
134              buffer ...
135          }
136      }
137      break; //Pokracujem v prijme dat ...
138      if(TI)
139      {
140          TI=0;
141          switch(++iout)
142          {
143              case preamblepkt
144              :
145                  //Tu by som sa nemal nikdy dostat,
146                  lebo som uz poslal Preamble ...
147                  case dstpkt : SOBUF=(TransmitMessage->Dst); break;
148                  case srcpkt : SOBUF=(TransmitMessage->Src); break;
149                  case sizeofpkt : SOBUF=(high(TransmitMessage->Size)); break;
150                  case sizeofpktlo : SOBUF=(low(TransmitMessage->Size)); break;
151                  case crcpkt : SOBUF=(TransmitMessage->Crc); break;
152                  case endpkt :
153                      //RtxCompletion=os_free_block(SizeMemBlock,MessageToSend); //RtxCompletion==NULL je OK ak
154                      uvolnim pamat ...
155                      send=OFF;
156                      break; //Nevymazavat, musi to tu byt !!!
157                  default : SOBUF=(TransmitMessage->Data[iout-headerpkt]);
158              }
159              break; //Poslem vsetky data ...
160          }
161          //Odosielanie znaku nemusim riesit, ide
162          prerusenim os_attach_interrupt(4) ...
163      }
164      }
165      }
166      }
167      void System(void) _task_ SYSTEM _priority_ 0
168      {
169          signed char RtxCompletion;
170          RtxCompletion=os_set_slice(15360);
171          RtxCompletion=os_create_pool(SizeMemBlock,MemoryPool,SizeMemPool); //Vytvorim MemoryPool v
172          pamati XRAM pre datove struktury ...
173          RtxCompletion=os_create_task(SERIAL);
174          RtxCompletion=os_create_task(COM);
175          while(1)
176          {
177              if(MessageToSend!=NULL) //Vytvorim packet ...
178              {
179                  Packet.Preamble=Token;
180                  Packet.Dst=(unsigned char)rand();
181                  Packet.Src=(unsigned char)rand();
182                  Packet.Size=sizeof(Packet.Data);
183                  strcpy(Packet.Data,"!!! Diversant Software !!!\r\n");
184                  Packet.Crc=Check_CRC((unsigned char xdata*)&Packet);
185                  memcpy(MessageToSend,&Packet,sizeof(Packet)); //Pamat je rezervovana a
186                  pozem kopirovat blok na odoslanie ...
187                  // while(recv==ON) RtxCompletion=os_wait(K_TMO,1,NULL); //Je prijem dat cez
188                  RS232 tak sa stop-nem na chvilu s vysielanim ...
189                  RtxCompletion=os_send_message(MEXSend,(unsigned int xdata*)MessageToSend,WaitForever);
190                  //Posielam adresu datovej struktury cez mailbox ...
191              }
192              // Packet.Crc=Check_CRC((unsigned char xdata*)MessageToSend); //Vypocitam CRC
193              // skopirovaného packetu a musia byt rovnake !!!
194              RtxCompletion=os_wait(K_TMO,250,NULL);
195          }
196      }
```



```
C:\Omega\Data\Dropbox\Záloha\C51\Multiprocessorová komunikácia s 8051\Multiproce
181 // RtxCompletion=os_free_block(SizeMemBlock,MessageToSend); //RtxCompletion==NULL je
    OK ak uvoľním pamäť v tasku SERIAL !!!
182 // os_delete_task(SYSTEM);
183 }
184 }
185
186 void main(void)
187 {
188     signed char RtxCompletion;
189     RtxCompletion=os_start_system(SYSTEM);
190 }
```

V praxi sa v súčasnej dobe čoraz častejšie stretávame s potrebou komunikácie viacerých mikroprocesorov navzájom. Situácia sa nám podstatne zjednodušuje integrovaným sériovým rozhraním RS232, CAN, I2C, Ethernet, prípadne iných komunikačných rozhraní. Sériový typ komunikácie je oproti paralelným formám podstatne jednoduchšie, pretože používa malý počet komunikujúcich vodičov. V praxi stačia na vytvorenie obojsmernej komunikácie 1 až 2 vodiče. Väčšina mikroprocesorov obsahuje integrované sériové komunikačné rozhranie RS232, preto sme sa rozhodli vytvoriť jednoduchý komunikačný model pre RS232. Ako základ sme si vzali technologicky vyspelejší a náročnejší Ethernet 802.3, ktorý je štandardom v oblasti počítačových sietí. Pre nás je podstatný samotný princíp komunikačného modelu. V tomto modeli má každý komunikujúci unikátnu MAC adresu ktorá ho jednoznačne identifikuje na fyzickej úrovni a IP adresu ktorá znova identifikuje zariadenie na logickej úrovni. K dátam ktoré sa odosielajú sa pripojí na začiatok dátovej oblasti blok ktorý obsahuje zdrojovú a cieľovú adresu. Na koniec tohoto bloku sa pripojí tzv. CRC kontrolný súčet, ktorý chráni prepravovaný blok dát pred poškodením pri transporte od zdroja k cieľu. Rovnaký model je použitý aj v tomto prípade. Zdrojová, cieľová adresa, veľkosť dátovej oblasti, CRC sú kvôli jednoduchej implementácii do 8051 len 8 bitové štruktúry. Podľa potreby je ich možné modifikovať. Takýmto spôsobom je možné vytvoriť komunikáciu nielen medzi dvojicou autonómnych systémov a dokážeme jednoznačne identifikovať zdroj a cieľ komunikácie.

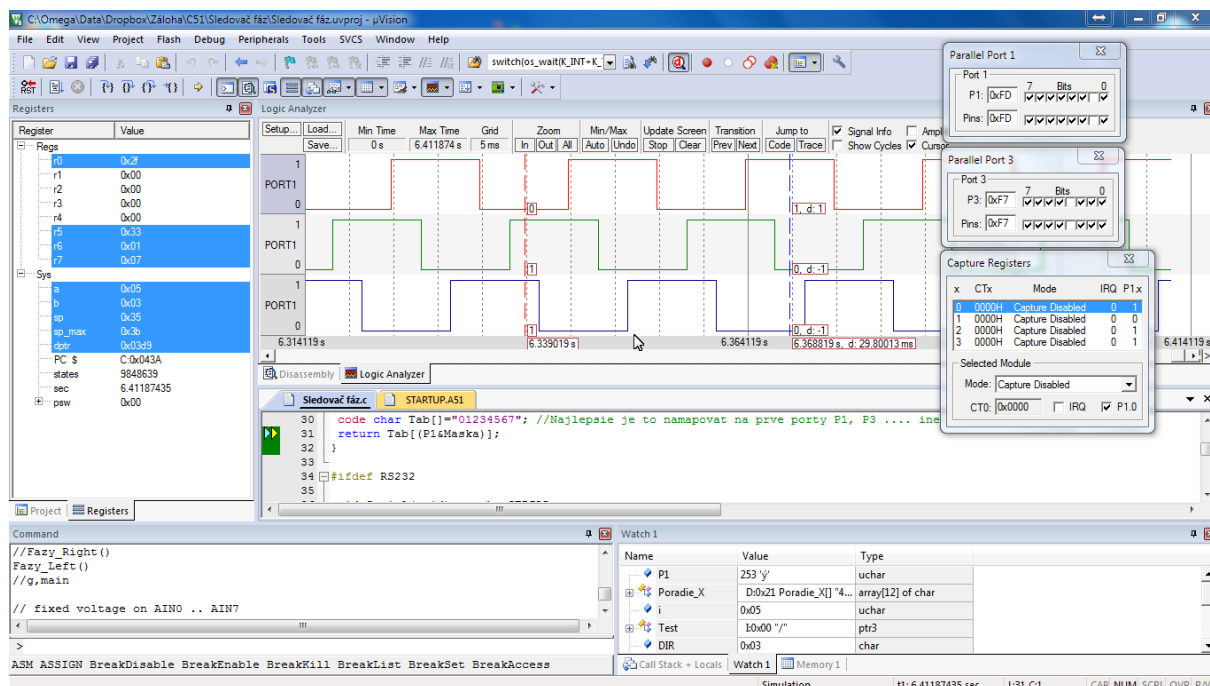
V našom prípade položku "SFD", "EtherType" nepoužijeme. Využijeme len "Preamble" identifikátor začiatku dátového paketu tzv. Tokenom "^" ktorý sme si zadefinovali ako štartovací znak, zdrojovú a cieľovú "MAC" adresu, dátovú oblasť "Payload" a kontrolný súčet "FCS" (Frame Check Sum) t.j. "CRC". Túto modifikovanú štruktúru budeme kvôli jednoduchosti programu odosielať pomocou funkcie **putchar()** a prijímať pomocou **getchar()**, prípadne vysielanie a príjem bude pre zvýšenie efektívnosti riadený prerušením. Celý program je napísaný v jazyku C s použitím RTX51 Full. S výhodou využíva oddelené prijímacie a vysielacie registre mikroprocesora **S0CON**, takže program je plne duplexný t.j. FDX.

Pamäťové štruktúry paketu (**Packet51**) sú staticky umiestnené v pamäťovom priestore **xdata**. Štruktúra **MessageToSend** a **ReceivedMessage** sú umiestnené do pamäti pomocou výkonných funkcií operačného systému RTX51 Full **os_get_block()**, ktorý je už alokovaný pomocou funkcie **os_create_pool()**. Odoslanie paketu sa uskutoční pomocou funkcie RTX51 **os_send_message()**. Možno sa niekomu môže zdať použitie funkcie **os_send_message()** ako zbytočné, ale funkcia obsahuje tzv. Buffer organizovaný ako (**FIFO First-In First-Out**) ktorý môže naraz zachytiť a spracovať až 8 správ v poradí ako prišli. Ako prvá je teda spracovaná tá správa ktorá čaká najdlhšie v rade, ktorá prišla ako prvá. V štruktúre **ReceivedRS232** t.j. **ReceivedRS232->Data** sú prijaté dáta cez RS232 na vlastnú adresu zariadenia. Štruktúra **TransmitMessage** t.j. **TransmitMessage->Data** obsahuje odosielané dáta na príslušnú adresu v sieti **232.NET**.

15.10. Elektronický sledovač fáz s mikroprocesorom riadený prerušením

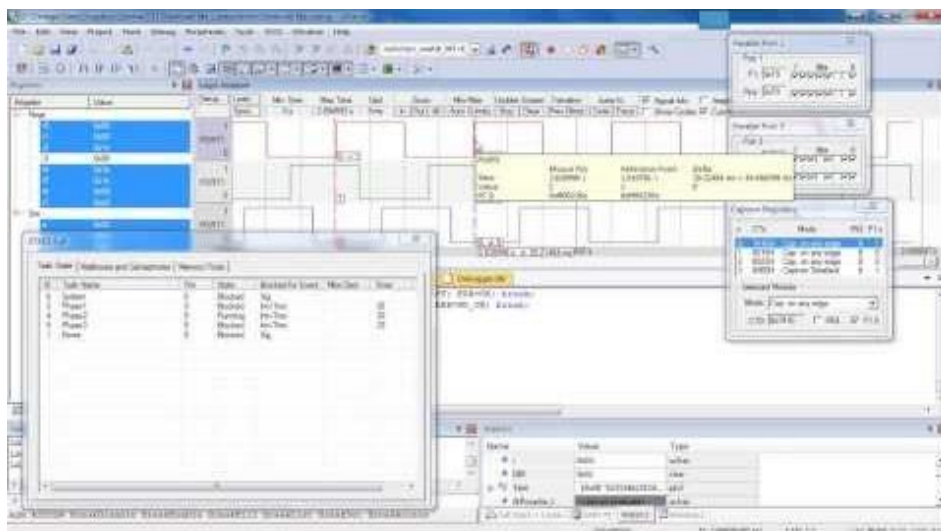
Srdcom elektronického sledovača fáz je mikroprocesor 80C552 na ktorý je privedená trojica signálov SL1 až SL3 ktoré predstavujú galvanicky oddelené signály TTL v 3-fázovej sieti. Tieto signály sú privedené na P1.0 až P1.2, ktoré sú pripojené interne na záchytnú jednotku CAPTURE mikroprocesora 80C552. Pripojenie na hardware záchytnej jednotky je vykonané trojicou task-ov s názvom Phase1 až Phase3 pomocou funkcie operačného systému RTX51 Full **os_attach_interrupt(n)**. Vyššie uvedené task-y sú nastavené na zmenu úrovne, takže pri zmene úrovne záchytná jednotka automaticky generuje prerušenie ktoré je signalizované bitom CT0 až CT2. Poradie fáz je uložené v tabuľke a v pravidelných intervaloch sú stavy na porte P1 porovnávané s hodnotami v tabuľke. V prípade výpadku fázy je do 20ms automaticky vygenerovaná chyba pomocou internej premennej ERR=0 ktorá je mapovaná do portu P3.2 a zároveň sa nastaví premenná DIR=0. V prípade správnej funkcie fáz je DIR=2 pre pravotočivý sled a DIR=3 pre ľavotočivý sled fáz. Rovnako sa nastaví premenné LEDR a LEDL mapované do portu P3.3 a P3.4, kde je sled signalizovaný nastavením⁹⁴ do logickej úrovne 1. Obrázok 110 zobrazuje zjednodušený variant sledovača je možné zostrojiť s jedným čipovým mikroprocesorom AT89LP4052..

Obrázok 109, Simulácia programu sledovača fáz

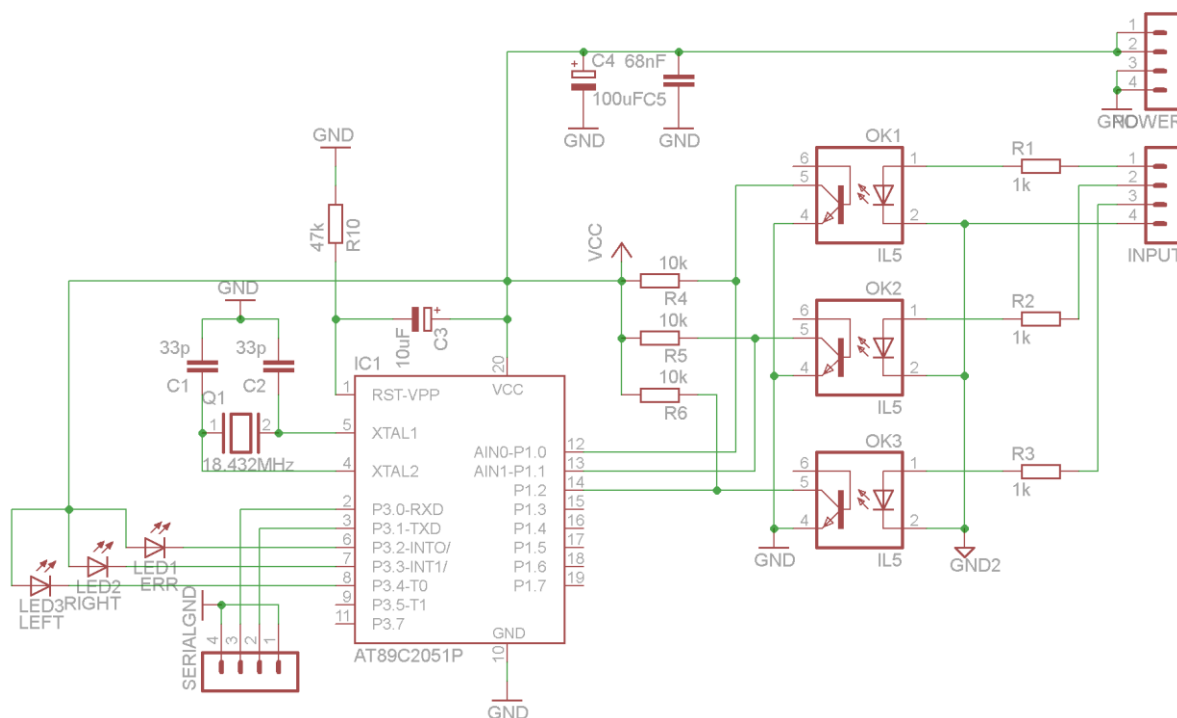


⁹⁴ Mikroprocesor 80C552 nemá posilňovač portu a maximálne je možné jeden vývod portu P1, alebo P3 zaťažiť prúdom približne I=1mA. Preto musíme použiť dodatočne na rozsvietenie príslušnej LED diódy TTL hradlo.

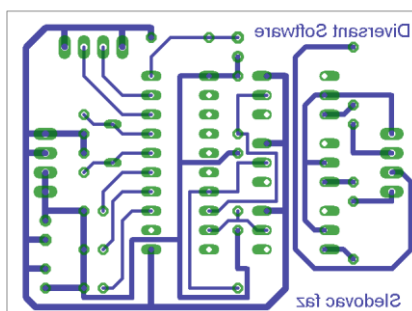
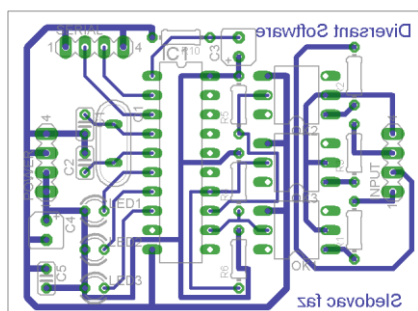
Video 1, Sledovač fáz s 80C552 riadený prerušením - simulácia



Obrázok 110, Schéma zapojenia sledovača fáz s AT89LP4052



Obrázok 111, Doska plošného spoja sledovača fáz s AT89LP4052



C:\Omega\Data\Dropbox\Záloha\C51\Sledovač fáz s prerušením\Sledovač fáz.c

```
1  #include <reg552.h>
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <string.h>
5  #include <rtx51.h>
6
7  #define MASK 0x07
8
9  #define SYSTEM 0
10 #define ERRORS 1
11 #define SERIAL 2
12 #define PHASE1 3
13 #define PHASE2 4
14 #define PHASE3 5
15
16 typedef enum {NO_OK=0, YES_OK=1, DIR_R=2, DIR_L=3} Chyby;
17 typedef enum {OFF=0, ON=1} Stav;
18 typedef enum {R=0, L=1} Smer;
19
20 char DIR; //Vysledok zistenia faz ...
21 sbit ERR = P3^2; //ERR=0 je chyba, ERR=1 je OK
22 sbit LEDR = P3^3; //LED signalizujuca otacanie sa faz RIGHT
23 sbit LEDL = P3^4; //LED signalizujuca otacanie sa faz LEFT
24
25 //123456 123456
26 code char *Smer[]={"645132","623154"};
27 data char Poradie_X[12];
28 code char *Message[]={"NOT_OK","OK","RIGHT","LEFT"};
29
30 unsigned char Port(unsigned char Maska)
31 {
32     code char Tab[]="01234567"; //Najlepsie je to namapovat na prve porty P1, P3 .... ine by boli
    komplikovanejsie ...
33     return Tab[(P1&Maska)];
34 }
35
36 void Serial(void) _task_ SERIAL
37 {
38     PCON|=0x80;
39     SOCON=0x50;
40     TMOD|=0x20;
41     TH1=TL1=0xFB;
42     TR1=1;
43     TI=1;
44     printf("System Ready ...\r\n");
45     while(1)
46     {
47         switch(os_wait(K_TMO+K_SIG,250,NULL))
48         {
49             case TMO_EVENT : printf("Smer otacania faz = %s\r\n",Message[DIR]);break;
50             case SIG_EVENT : printf("Chyba %02bd\r\n",DIR); break;
51         }
52     }
53 }
54
55 void Errors(void) _task_ ERRORS
56 {
57     ERR=OK;
58     while(1)
59     {
60         switch(os_wait(K_SIG,0xFF,NULL))
61         {
62             case SIG_EVENT : ERR=NO_OK;
63                             DIR=NO_OK;
64                             os_send_signal(SERIAL);
65                             break;
66             case TMO_EVENT : ERR=YES_OK; break; //Nulujem WDT, inac by to skoncil
67         }
68     }
69 }
70
71 unsigned char i = 0;
72
73 void Reload(void)
```

Page 1

C:\Omega\Data\Dropbox\Záloha\C51\Sledovač fáz s prerušením\Sledovač fáz.c

```
74 {
75     if(i>=sizeof(Poradie_X))
76     {
77         i=0x00;
78         os_send_signal(SYSTEM);
79     }
80 }
81
82 void Phase1(void) _task_ PHASE1 _priority_ 0
83 {
84     TM2CON=0x01;
85     CTCON|=0x02+0x01;
86     os_attach_interrupt(6);
87     while(1)
88     {
89         switch(os_wait(K_INT+K_TMO,30,NULL)) //Cakam na fazu A a zachytavam obsahy
90             zachytnych registrov ktore vyjadruju dlzku 1/2 periody fazoveho napatia siete
91         {
92             case INT_EVENT : CTI0=0;
93                             Reload();
94                             Poradie_X[i++]=Port(MASK); break; //Prisla hrana fazy, tak vykonam
95             toto ...
96             case TMO_EVENT : os_send_signal(ERRORS); break; //Ak sa tu dostanem neprisla hrana
97             fazy do TIME tak vyvolam chybu ...
98         }
99     }
100 }
101 void Phase2(void) _task_ PHASE2 _priority_ 0
102 {
103     TM2CON=0x01;
104     CTCON|=0x08+0x04;
105     os_attach_interrupt(7);
106     while(1)
107     {
108         switch(os_wait(K_INT+K_TMO,30,NULL)) //Cakam na fazu B a zachytavam obsahy
109             zachytnych registrov ktore vyjadruju dlzku 1/2 periody fazoveho napatia siete
110         {
111             case INT_EVENT : CTI1=0;
112                             Reload();
113                             Poradie_X[i++]=Port(MASK); break; //Prisla hrana fazy, tak vykonam
114             toto ...
115             case TMO_EVENT : os_send_signal(ERRORS); break; //Ak sa tu dostanem neprisla hrana
116             fazy do TIME tak vyvolam chybu ...
117         }
118     }
119 }
120 void Phase3(void) _task_ PHASE3 _priority_ 0
121 {
122     TM2CON=0x01;
123     CTCON|=0x20+0x10;
124     os_attach_interrupt(8);
125     while(1)
126     {
127         switch(os_wait(K_INT+K_TMO,30,NULL)) //Cakam na fazu C a zachytavam obsahy
128             zachytnych registrov ktore vyjadruju dlzku 1/2 periody fazoveho napatia siete
129         {
130             case INT_EVENT : CTI2=0;
131                             Reload();
132                             Poradie_X[i++]=Port(MASK); break; //Prisla hrana fazy, tak vykonam
133             toto ...
134             case TMO_EVENT : os_send_signal(ERRORS); break; //Ak sa tu dostanem neprisla hrana
135             fazy do TIME tak vyvolam chybu ...
136         }
137     }
138 }
139 void System(void) _task_ SYSTEM _priority_ 0
140 {
141     char *Test;
142     os_set_slice(1536);
143     os_wait(K_TMO,20,NULL);
144     #ifdef RS232
145     os_create_task(SERIAL);
146     }
```

Page 2

C:\Omega\Data\Dropbox\Záloha\C51\Sledovač fáz s prerušením\Sledovač fáz.c

```
140  #endif
141  os_create_task(PHASE1);
142  os_create_task(PHASE2);
143  os_create_task(PHASE3);
144  os_create_task(ERRORS);
145  while(1)
146  {
147      os_wait(K_SIG, 0xFF, NULL);
148      if((Test=strstr(Poradie_X, Smer[R]))!=NULL) DIR=DIR_L; //Smer otacania sa faz je vlavo
149      if((Test=strstr(Poradie_X, Smer[L]))!=NULL) DIR=DIR_R; //Smer otacania sa faz je vpravo
150      switch(DIR)
151      {
152          case DIR_R : LEDR=ON; LEDL=OFF; ERR=OK; break;
153          case DIR_L : LEDL=ON; LEDR=OFF; ERR=OK; break;
154          default    : LEDL=LEDR=OFF; ERR=NO_OK; break;
155      }
156  }
157 }
158
159 void main(void)
160 {
161     os_start_system(SYSTEM);
162 }
```

15.11. Elektronický sledovač fáz s mikroprocesorom a priamym meraním frekvencie

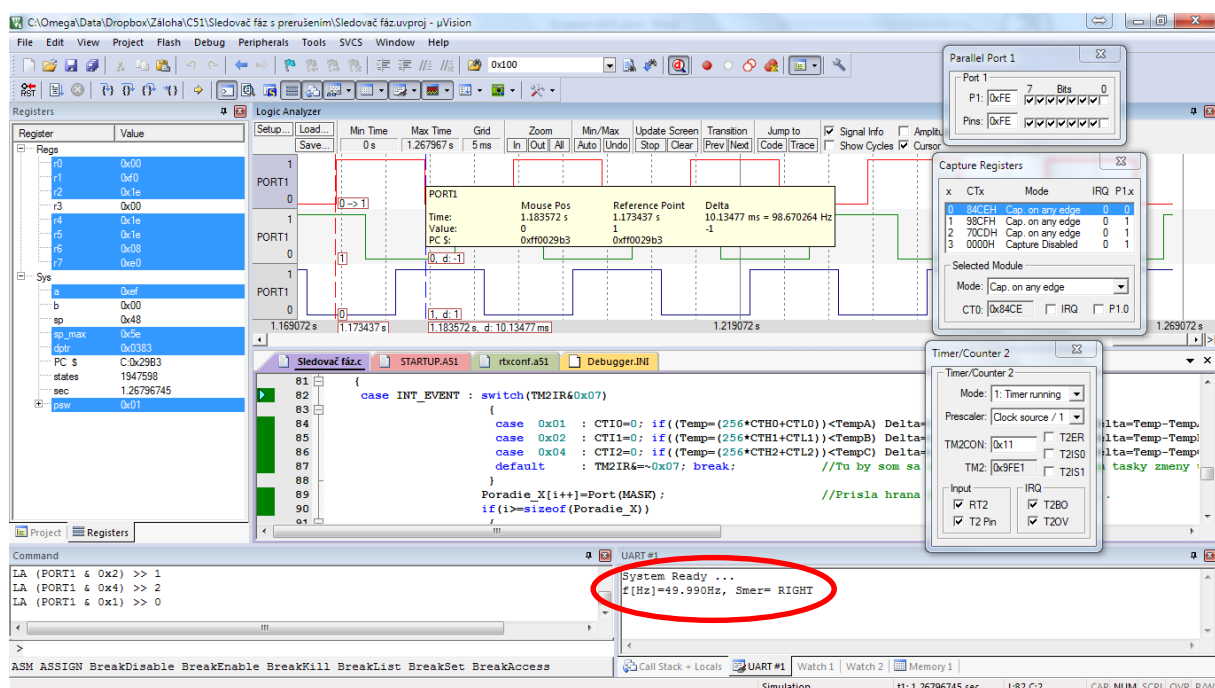
Konštrukcia vyššie uvedeného elektronického zariadenia je rovnaká ako v kapitole 15.10, pričom sa upravil len samotný podprogram⁹⁵ obsluhy prerušenia záchytnej jednotky CAPTURE. V premennej **Delta** je uložený počet impulzov medzi dvomi prerušeniami jednej záchytnej jednotky a vyjadruje jednu polovicu periódy sieťového napätia. Záchytná jednotka je v systéme 80C552 napájaná frekvenciou $f_{CAPTURE} = \frac{1}{12} \cdot f_{XTAL}$ t.j. $\frac{1}{12}$ frekvencie oscilátora.

Výslednú veľkosť frekvencie napätia v sieti je možné vypočítať ako:

$$f = \frac{f_{XTAL}}{24 \cdot (256 \cdot CTH0 + CTL0)} \quad [\text{Hz}; \text{Hz}, 1, 1] \quad (6.)$$

f_{XTAL} je frekvencia kryštálu použitého v konštrukcii mikroprocesorovej jednotky
 $CTH0$ a $CTL0$ sú registre mikroprocesora záchytnej jednotky CAPTURE

Obrázok 112, Priame meranie frekvencie mikroprocesorom 80C552 riadený prerušením



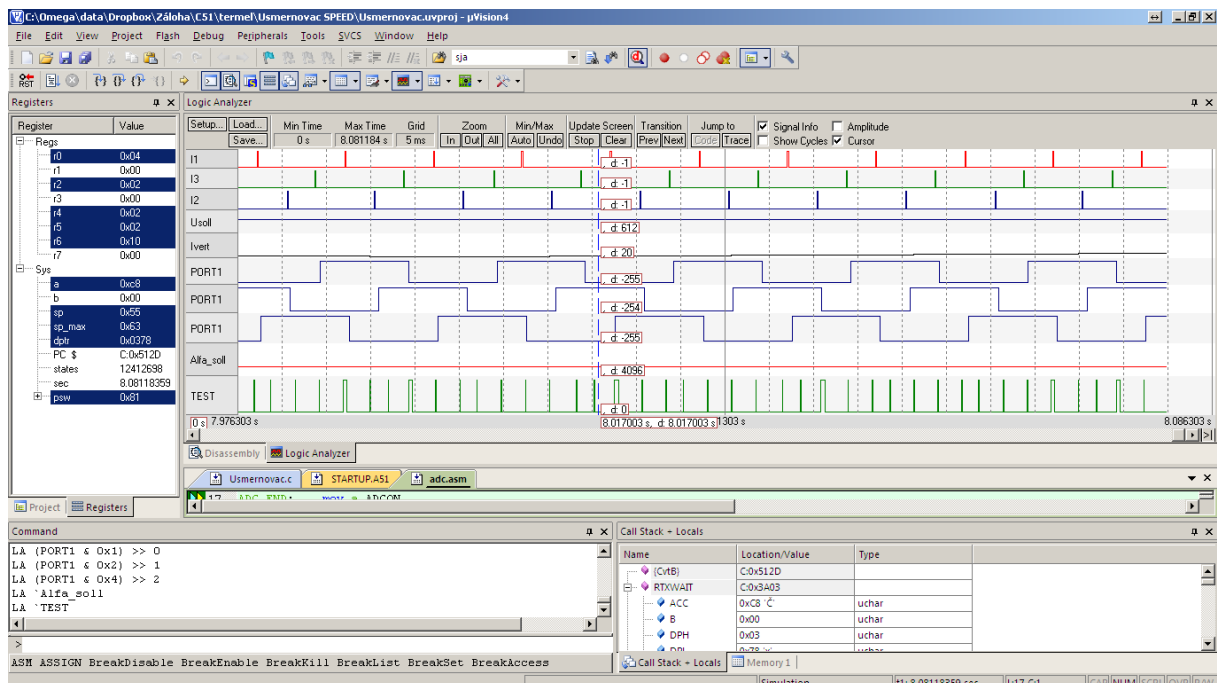
⁹⁵ Odporúčanie pre prax: Task **PHASE** by mal mať najvyššiu úroveň priority (1 až 3), aby sa prednostne a načas zabezpečila obsluha prerušení od záchytnej jednotky. V opačnom prípade by dochádzalo k nesprávnym meraniam a tým skresleniu výsledných hodnôt.

15.12. Tyristorový regulátor – fázové riadenie usmerňovača

C:\Omega\data\Dropbox\Záloha\C51\termel\Usmernovac\Usmernovac.c

```
233 #pragma REGISTERBANK (3)
234
235 sbit I1 = P4^0;
236 sbit I3 = P4^1;
237 sbit I2 = P4^2;
238
239 void Impuls(void) _task_ IMPULS _priority_ 3
240 {
241     #define Length 75 //60*1.366=85us pre dlzku
242     #define Dlzka (Length/1.366) //Aby som nemusel stale
243     //prepocitavat na skutocnu dlzku impulsu ...
244     os_attach_interrupt(11);
245     os_attach_interrupt(12);
246     os_attach_interrupt(13);
247     while(1)
248     {
249         Watchdog ;
250         switch (os_wait (K_TMO+K_INT,K*100, NULL))
251         {
252             case INT_EVENT : switch (TM2IR & 0x70) //RTX51 HANDLER prepne do cca.
253             //110us na tento task zovsadiat !!!
254             {
255                 case 0x10 : CMI0=0; I1=1; Delay (Dlzka); I1=0; break ; //Generujem 100us
256                 impuls I1 pre SL1 pre podmienku CMH0+CML0=TMH2+TML2
257                 case 0x20 : CMI1=0; I3=1; Delay (Dlzka); I3=0; break ; //Generujem 100us
258                 impuls I3 pre SL3 pre podmienku CMH1+CML1=TMH2+TML2
259                 case 0x40 : CMI2=0; I2=1; Delay (Dlzka); I2=0; break ; //Generujem 100us
260                 impuls I2 pre SL2 pre podmienku CMH2+CML2=TMH2+TML2
261                 default : TM2IR &= 0x70 ; break ; //Ak som tu tak z
262                 //nevysvetliteľných dôvodov prisli niektoré, alebo všetky dve-tri prerušenia od komparátora !!!
263                 //Stane sa to pri poruchách na silových fázach a vtedy nesmie generovať impulz !!! ...
264             }
265             break ;
266             case TMO_EVENT : os_send_message (ERROR_MAIL,IMPULS,TIME); break ;
267         }
268     }
269 }
```

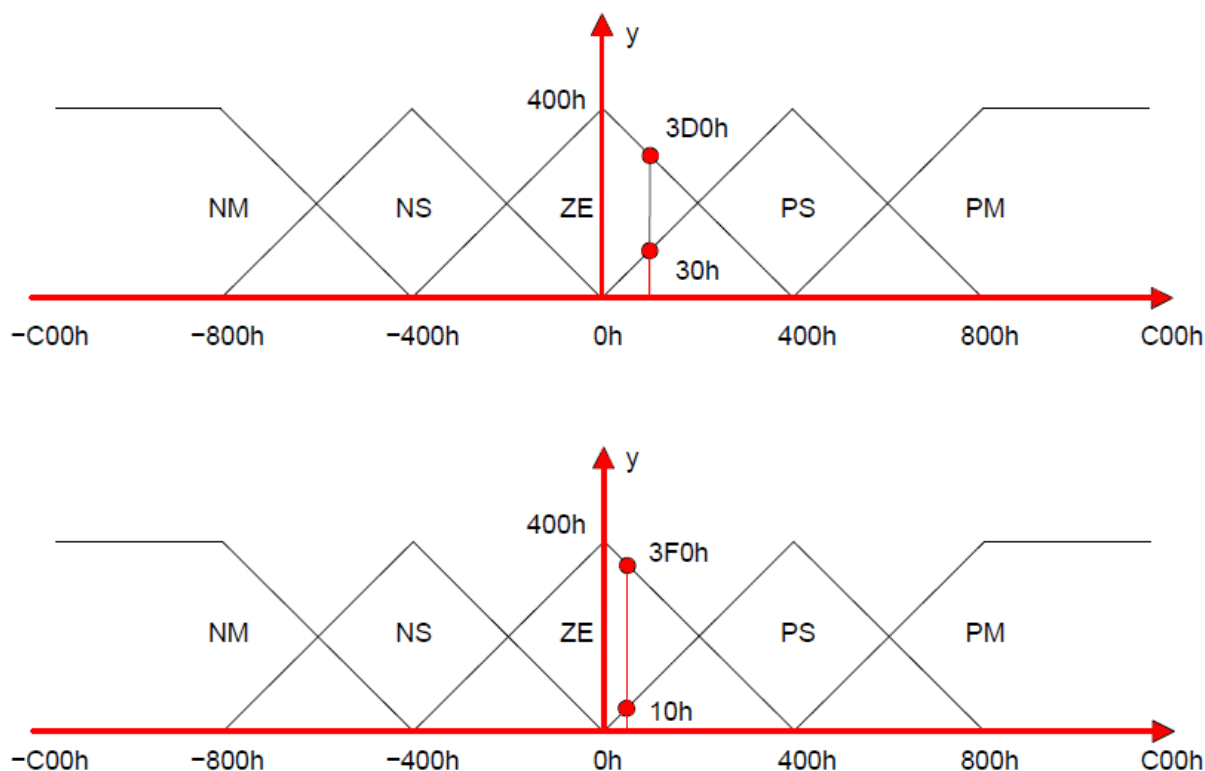
Obrázok 113, Generovanie zapínacích impulzov - simulácia



15.13. Fuzzy regulátor v C

Teória Fuzzy regulácie je známa približne od roku 1980. Najväčší pokrok v tejto oblasti dosiahli Japonci, ktorí sa snažili v čo najväčšej miere implementovať jej princípy do praxe. V princípe ide o regulovanie procesu na základe slovného opisu problému. Najlepším príkladom je napríklad šoférovanie v autoškole. Inštruktor v autoškole neučí žiaka tak, že mu prikáže pri mierne pravotočivej zákrute otočiť volant riadenia o $35,6^\circ$ vpravo, ale prikáže mu pootočiť volant mierne vpravo. Pri prudkej zákrute mu prikáže vytočiť volant na pravý doraz doprava. To sú zjednodušené princípy Fuzzy regulácie. Fuzzy algoritmy nie sú extra matematicky zložité a pre navrhovanie úplne stačia aj stredoškolské znalosti matematiky. Tieto algoritmy bývajú veľmi robustné a odolné proti rôznym chybám hodnôt a šumom. Ich veľkou nevýhodou je, že vďaka tomu môžu byť niekedy pomerne nepresné. Ak sa chceme tomu vyhnúť tak je ich potrebné spojiť s lineárnym regulátorom. Bližšie informácie môžete nájsť v (Novák, 2000).

Obrázok 114, Zaradenie hodnoty k príslušnosti prvku danej množiny



Obrázok 115, Fuzzy regulátor v jazyku C s RTX51

C:\Omega\data\Dropbox\Záloha\C51\fuzzy\Fuzzy.c

```
1  //*****
2  // Header: FUZZY regulator s RTX51 pre taviace pece
3  // File Name: FUZZY.C
4  // Author: Ing. Eduard Jadron, Jilemnického 3999/37, 03601 Martin, Slovak Republic
5  // Date: 29.12.2005
6  //*****
7
8  #include <reg552.h>
9  #include <stdio.h>
10 #include <stdlib.h>
11 #include <rtx51.h>
12 #include <dsw.h>
13
14 #define REGULATOR 0
15 #define ACTION 1
16
17 int Error,dError,LastError;
18
19 #define SIZE 5 //Velkost inferencneho bloku !!!
20
21 int X1[SIZE],X2[SIZE],Y[SIZE],Output;
22
23 int Soll,Vert;
24
25 //typedef enum (NM,NS,ZE,PS,PM) State;
26
27 #define NM 0 // Negative medium
28 #define NS 1 // Negative small
29 #define ZE 2 // Zero equal
30 #define PS 3 // Positive small
31 #define PM 4 // Positive medium
32
33 const unsigned char InferenceTable[SIZE][SIZE] =
34 {
35     /*NM,NS,ZE,PS,PM*/ //dError X2[]
36     /*NM*/ {EM,PM,PM,PS,ZE}, //NM E
37     /*NS*/ {EM,PM,PS,ZE,NS}, //NS r
38     /*ZE*/ {EM,PS,ZE,NS,NM}, //ZE r
39     /*PS*/ {PS,ZE,NS,NM,NM}, //PS o
40     /*PM*/ {ZE,NS,NM,NM,NM} //PM r
41 };
42
43 #define K 0x18
44
45 const int OutputFunc[SIZE] = {-2*K,-K,+0x00,+K,+2*K};
46
47 unsigned int ADC(unsigned char Channel) //Rutina trva 55us pri 18.432MHz
48 {
49     #define ADCS 0x08
50     #define ADCI 0x10
51     #define ADEX 0x20
52     unsigned int x;
53     ADCON = Channel; //Vyber kanalu pre prevod 0..7
54     ADCON |= ADEX + ADCS; //Spustim prevod, bolo tu ADCON|=ADEX+ADCS;
55     while ((ADCON & ADCI) == 0x00); //Cakam na ukoncenie prevodu
56     x = ADCCH * 0x04; //Nulujem priznak AD prevodnika
57     switch (ADCON & (0x80 + 0x40)) //Normovanie na (int)
58     {
59         case 0x40 : x += 0x01; break;
60         case 0x80 : x += 0x02; break;
61         case 0xC0 : x += 0x03; break;
62     }
63     return (x);
64 }
65
66 typedef enum {SCALE=0x0400,SK0=+0x0000,SK1=+0x0400,SK2=+0x0800} Osi;
67
68 void Fuzzification(int Value, int *Data) //reentrant
69 {
70     unsigned char i;
71     for(i=0x00; i<SIZE; i++) Data[i]=0x00;
72     if(Value <= SK2) Data[NM]=SCALE;
```

C:\Omega\data\Dropbox\Záloha\C51\fuzzy\Fuzzy.c

```
73     else if (Value <= SK1 )
74     {
75         Data [NM]= SCALE -(Value +SK2 );
76         Data [NS]=Value +SK2 ;
77     }
78     else if (Value <= SK0 )
79     {
80         Data [NS]= SCALE -(Value +SK1 );
81         Data [ZE]=Value +SK1 ;
82     }
83     else if (Value <= SK1 )
84     {
85         Data [ZE]= SCALE -Value ;
86         Data [PS]=Value ;
87     }
88     else if (Value <= SK2 )
89     {
90         Data [PS]= SCALE -(Value -SK1 );
91         Data [PM]=Value -SK1 ;
92     }
93     else Data [PM]= SCALE ;
94 }
95
96 void FuzzyInference (int *X1, int *X2, int *Y) //reentrant
97 {
98     int min [SIZE];
99     int max ,Temp ;           //Tu musi byt v pripade RTX51 a LARGE modelu "data int max" s
100    optimalizaciou Loop rotation - 6
101    unsigned char i,j,maxpos ;
102    for (i=0x00 ; i<SIZE; i++) Y[i]= 0x00 ; //Vymazem vystupny vektor Y[]
103    for (i=0x00 ; i<SIZE; i++)
104    {
105        for (j=0x00 ; j<SIZE; j++)
106            if (X1 [i]<X2 [j]) min [j]=X1 [i]; else min [j]=X2 [j];
107        max =min [0];
108        maxpos =0x00 ;
109        for (j=0x01 ; j<SIZE; j++)
110            if (max <min [j])
111            {
112                max =min [j];
113                maxpos =j;
114            }
115        Temp =Y[InferenceTable [i][maxpos ]];
116        if (Temp <max) Y[InferenceTable [i][maxpos ]]+= max ;
117        if (Temp >SCALE) Y[InferenceTable [i][maxpos ]]= SCALE ;
118    }
119
120    int Defuzzification (int *Y) //reentrant
121    {
122        unsigned char i;
123        int ReturnVal =0,SumY=0;
124        for (i=0x00 ; i<SIZE; i++)
125        {
126            SumY +=Y[i];
127            ReturnVal +=Y[i]* OutputFunc [i];
128        }
129        return ((long) ReturnVal <<2)/ SumY ; //Skalujem cislom 4
130    }
131
132    void Regulator (void) _task_ REGULATOR _priority_ 0
133    {
134        os_set_slice (xtal (18.432E6)/10);
135        PWMp=0x20 ;
136        Soll =Vert =0x0000 ;
137        while (1)
138        {
139            // Vert=ADC (0);
140            if (++ Vert >= 1024 ) Vert =0;
141            Soll =ADC (2);
142            LastError =Error ;
143            Error =(Vert -Soll) << 1;
144            dError =(Error -LastError);
145            Fuzzification (Error ,X1);
```

Page2

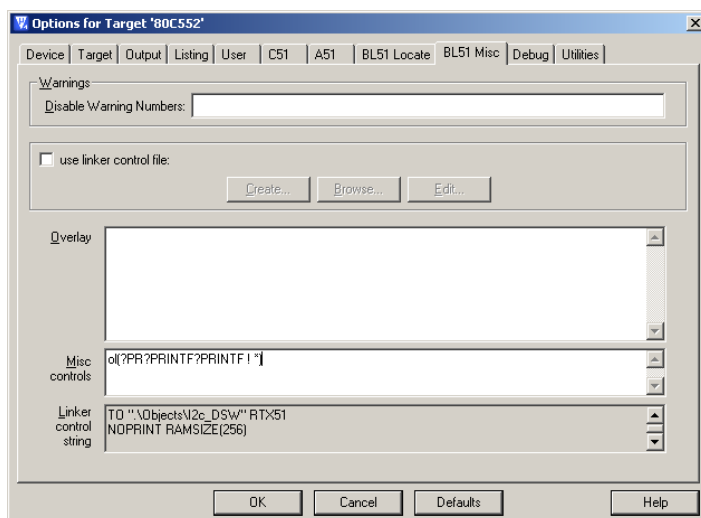
C:\Omega\data\Dropbox\Záloha\C51\fuzzy\Fuzzy.c

```
146     Fuzzification (dError ,X2);
147     FuzzyInference (X1,X2,Y);
148     Output =Defuzzification (Y);
149     //   PWM0=0x80-Output;
150     PWM0 =PWM1 =/*0x80-*/ Output ;
151     os_wait (K_TMO ,2,NULL);
152 }
153 }
154
155 void main (void)
156 {
157     os_start_system (REGULATOR);
158 }
```

15.14. Použitie „overlay“

RTX51 a každý operačný systém je nesmierne mocným nástrojom ktorý umožňuje programátorovi vytvoriť vysoko efektívny programový kód. Ak si uvedomíme, že sa jedná o prostredie, kde je možné spustiť viac procesov súčasne, čoskoro nastane problém so súčasným prístupom z viacerých úloh t.j. taskov k jednej funkcii, alebo procedúre. Najlepším príkladom je práve funkcia **printf()**. Neraz sa vyskytne v praxi požiadavka vypísať práve pomocou funkcie **printf()** do prenosového kanálu z dvoch úloh súčasne. Táto funkcia nie je reentrant-ná a tak jej spustenie z dvoch úloh nie je možné, pretože sa prepíšu jej interné premenné, čo spôsobí jej nekorektnú funkciu. Obrázok 116 ukazuje ako je možné využívať rovnakú funkciu z viacerých úloh v spolupráci so semaformi.

Obrázok 116, Použitie a zápis overlay-u v µVision



15.15. Použitie „overlay“ direktívy pri zdieľaní funkcií

Ak niektorá používaná funkcia v prostredí RTX51 nie je reentrant-ná, tak jej súčasné spustenie z dvoch rôznych úloh spôsobí nesprávnu funkciu. Obrázok 117 názorne ukazuje ako je možné využívať rovnakú funkciu z viacerých úloh v spolupráci so semaformi a ukazuje jednoduchý príklad ošetrovania zápisu súčasného prístupu⁹⁶ z viacerých task-ov na jednu funkciu. Pritom je nutné zabezpečiť, aby nedošlo k prepnutiu z jednej úlohy vykonávajúcou rovnakú funkciu druhou úlohou vykonávajúcou tú istú funkciu. Tento problém je možné riešiť pomocou semaforov. Práve binárny semafor zabezpečí, aby súčasne do rovnakej funkcie ktorá nie je

⁹⁶ V prípade reentrant-ných funkcií, alebo procedúr ktoré majú k dispozícii alokovanú dostatočne veľkú vlastnú pamäť pre „vnorenie“ by nemalo dôjsť k nesprávnej funkcii pri súčasnom využívaní jednej funkcie viacerými úlohami.

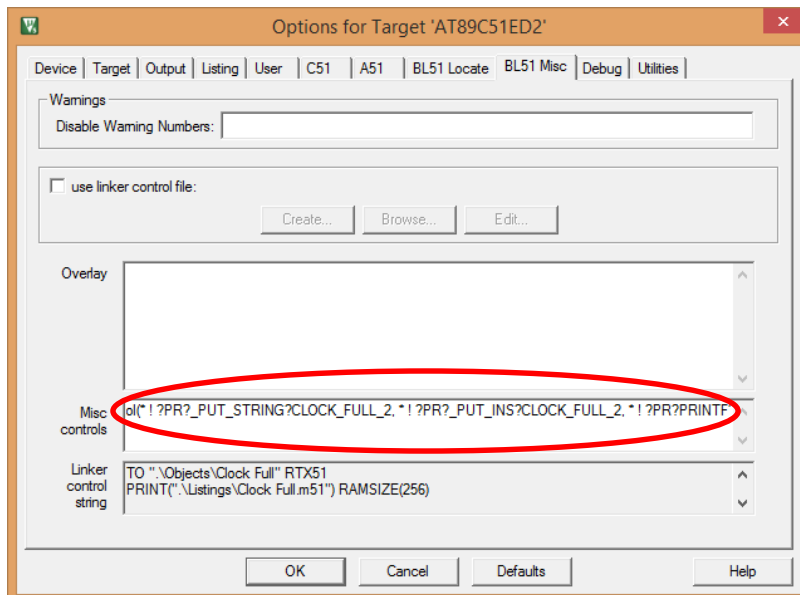
reentrant-ná nepokúsili sa prístupit' viaceré úlohy a nespôsobili tak prípadné prepísanie environment-u samotnej funkcie. V prípade že sa tak stane, „prepnutá“ funkcia bude obsahovať neplatné hodnoty. Linker μ Vision pri súčasnom prístupe viacerých task-ov do jednej funkcie vypíše varovné hlásenie, ktoré musíme ošetriť použitím štandardných direktív.

Príklad zápisu overlay-u pri viacnásobnom použití jednej funkcie rôznymi task-mi v RTX51:

- `ol(?PR?_PUT_STRING?CLOCK_FULL_2 !*,?PR?_PUT_INS?CLOCK_FULL_2 !*,?PR?PRINTF?PRINTF !*)`
- `ol(*! ?PR?_PUT_STRING?CLOCK_FULL_2, *! ?PR?_PUT_INS?CLOCK_FULL_2, *! ?PR?PRINTF?PRINTF)97`

Z vyššie uvedeného je vidieť, že v zdrojovom súbore s názvom **clock_full_2.c** je opakovane volaná funkcia **put_string()**, **put_ins()** a **printf()** z viacerých task-ov.

Obrázok 117, Ošetrovanie viacnásobne volaných funkcií v μ Vision5



⁹⁷ Rovnaký zápis, len iným spôsobom overlay direktívy uvedený BL51Misc v Options for Target a položke Misc Controls.

Obrázok 118, Chybové hlásenia pri súčasnom prístupe viacerých úloh

```
Rebuild target 'AT89C51ED2'
compiling Clock Full 2.c...
assembling EE_ASM.ASM...
assembling RTXCONF.A51...
assembling STARTUP.A51...
assembling KALENDAR.ASM...
linking...
*** WARNING L15: MULTIPLE CALL TO SEGMENT
SEGMENT: ?PR?PRINTF?PRINTF
CALLER1: ?PR?DISPLAY?CLOCK_FULL_2
CALLER2: ?PR?KEYBOARD?CLOCK_FULL_2
*** WARNING L15: MULTIPLE CALL TO SEGMENT
SEGMENT: ?PR?_PUT_STRING?CLOCK_FULL_2
CALLER1: ?PR?DISPLAY?CLOCK_FULL_2
CALLER2: ?PR?KEYBOARD?CLOCK_FULL_2
*** ERROR L107: ADDRESS SPACE OVERFLOW
SPACE: XDATA
SEGMENT: ?RTX?TASKCONTEXT?10
LENGTH: 0032H
Program Size: data=96.5 xdata=1789 code=18907
Target not created.
```

Obrázok 119, Výpis linkera L51 po kompilácii programu s použitím overlay direktív

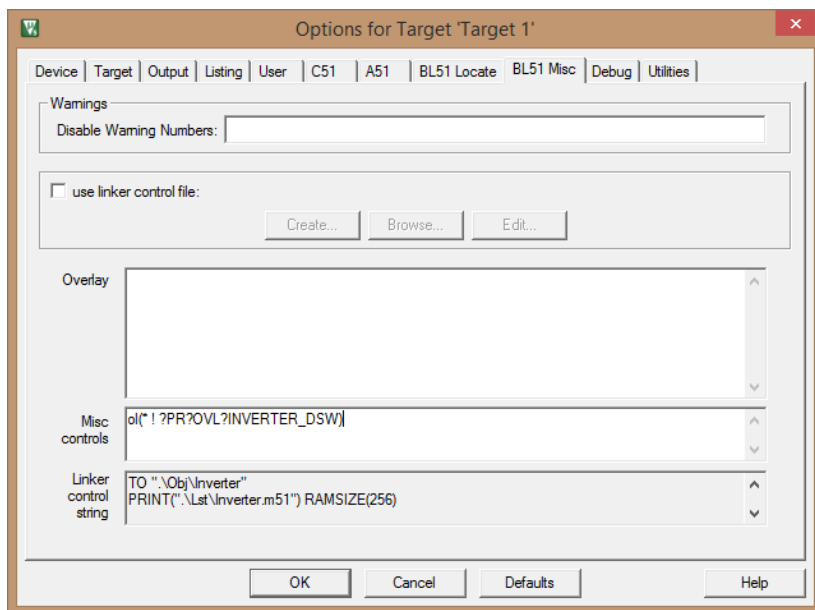
```
Rebuild target 'AT89C51ED2'
compiling Clock Full 2.c...
assembling EE_ASM.ASM...
assembling RTXCONF.A51...
assembling STARTUP.A51...
assembling KALENDAR.ASM...
linking...
Program Size: data=90.4 xdata=1764 code=18907
creating hex file from ".\Objects\Clock Full"...
".\Objects\Clock Full" - 0 Error(s), 0 Warning(s).
```

V praxi sa môže stať, že sa v programe nachádza funkcia, ktorá je deklarovaná, ale nie je volaná. Programátor môže ošetriť tento stav pomocou „overlay“ direktívy, čím sa odstráni zbytočné chybové hlásenie s kódom L16⁹⁸.

Funkcia, ktorá nie je volaná z hlavného programu spôsobí generovanie chybového hlásenia, ktoré je možné potlačiť použitím direktívy overlay-a `ol(* ! ?PR?_MENO_FUNKCIE)`, kde ?PR? je určenie segmentu v ktorom sa nachádza funkcia za ktorú nasleduje meno funkcie.

⁹⁸ *** WARNING L16: UNCALLED SEGMENT, IGNORED FOR OVERLAY PROCESS SEGMENT: ?PR?MYFUNC?X. Uvedené chybové hlásenie je signálom, že uvedená funkcia, alebo procedúra ktorá je deklarovaná v programe nie je vôbec volaná. Toto chybové hlásenie je možné jednoducho programátorsky odstrániť vloženíím „overlay“ direktívy prostredia µVision5 `ol(* ! ?PR?NAME_FUNCTION?NAME_SOURCE_FILE)` do BL51Misc v okne Options for Target. Chybové hlásenie sa už nebude pri preklade programu vypisovať a linker potvrdzuje, že preklad programu bol úspešný.

Obrázok 120, Príklad zápisu ošetrenia nevolanej funkcie



Príklad: v súbore **inverter_dsw.c** je deklarovaná funkcia **(void)Ovl(void)**, ktorá v celom programe neobsahuje funkčné volanie. Na odstránenie chybového hlásenia o nevolanom segmente použijeme direktívu **ol(* ! ?PR?OVL?INVERTER_DSW)**⁹⁹, ktorá automaticky zabezpečí, že linker nebude generovať chybové hlásenie.

Obrázok 121, Výpis programu s chybovým hlásením o nevolanom segmente

```
Rebuild target 'Target 1'
assembling STARTUP.A51...
compiling inverter_dsw.c...
assembling inverter_asm.asm...
linking...
*** WARNING L16: UNCALLED SEGMENT, IGNORED FOR OVERLAY PROCESS
    SEGMENT: ?PR?OVL?INVERTER_DSW
Program Size: data=12.0 xdata=0 code=363
creating hex file from ".\Obj\Inverter"...
".\Obj\Inverter" - 0 Error(s), 1 Warning(s).
```

Obrázok 122, Výpis programu bez chybového hlásenia o nevolanom segmente

```
Rebuild target 'Target 1'
assembling STARTUP.A51...
compiling inverter_dsw.c...
assembling inverter_asm.asm...
linking...
Program Size: data=12.0 xdata=0 code=363
creating hex file from ".\Obj\Inverter"...
".\Obj\Inverter" - 0 Error(s), 0 Warning(s).
```

⁹⁹ Je možné použiť aj direktívu s veľkými písmenami **OL(* ! ?PR?OVL?INVERTER_DSW)**

15.16. Ovládač RS232 s využitím I/O buffer-a riadeného prerušením

c:\Keil\C51\Rtx51\Examples\Trafic\SERIAL.C

```
31  /*****
32  /*      putchar: write a character to SBUF or transmission buffer      */
33  /*****
34  putchar (char c) {
35      if (!sendfull) {          /* transmit only if buffer not full */
36          if (!sendactive && !sendstop) { /* if transmitter not active: */
37              sendactive = 1;          /* transfer the first character direct*/
38              SBUF = c;                /* to SBUF to start transmission */
39          }
40          else {                /* otherwise: */
41              outbuf[oend++ & (OLEN-1)] = c; /* transfer char to transmission buffr*/
42              if (((oend ^ ostart) & (OLEN-1)) == 0) sendfull = 1;
43          }                      /* set flag if buffer is full */
44      }
45  }
46  /*****
47  /*      putchar: interrupt controlled putchar function */
48  /*****
49  char putchar (char c) {
50      if (c == '\n') {          /* expand new line character: */
51          while (sendfull) {    /* wait for transmission buffer empty */
52              otask = os_running_task_id (); /* set output task number */
53              os_wait (K_SIG, 0, 0);        /* RTX-51 call: wait for signal */
54              otask = 0xff;              /* clear output task number */
55          }
56          putchar (0x0D);        /* send CR before LF for <new line> */
57      }
58      while (sendfull) {        /* wait for transmission buffer empty */
59          otask = os_running_task_id ();    /* set output task number */
60          os_wait (K_SIG, 0, 0);            /* RTX-51 call: wait for signal */
61          otask = 0xff;                    /* clear output task number */
62      }
63      putchar (c);              /* send character */
64      return (c);              /* return character: ANSI requirement */
65  }
66  /*****
67  /*      _getkey: interrupt controlled _getkey */
68  /*****
69  char _getkey (void) {
70      while (iend == istart) {
71          itask = os_running_task_id ();    /* set input task number */
72          os_wait (K_SIG, 0, 0);            /* RTX-51 call: wait for signal */
73          itask = 0xff;                    /* clear input task number */
74      }
75      return (inbuf[istart++ & (ILEN-1)]);
76  }
77  /*****
78  /*      serial: serial receiver / transmitter interrupt */
79  /*****
80  serial () interrupt 4 using 1 { /* use registerbank 1 for interrupt */
81      unsigned char c;
82      bit start_trans = 0;
83      if (RI) { /* if receiver interrupt */
84          c = SBUF; /* read character */
85          RI = 0; /* clear interrupt request flag */
86          switch (c) { /* process character */
87              case CTRL_S: /* if Control+S stop transmission */
88                  sendstop = 1;
89                  break;
90
91              case CTRL_Q: /* if Control+Q start transmission */
92                  start_trans = sendstop;
93                  sendstop = 0;
94                  break;
95
96              default: /* read all other characters into inbuf*/
97                  if (istart + ILEN != iend) {
98                      inbuf[iend++ & (ILEN-1)] = c;
99                  }
100                  /* if task waiting: signal ready */
101                  if (itask != 0xFF) isr_send_signal (itask);
102                  break;
103          }
104      }
105      if (TI || start_trans) { /* if transmitter interrupt */
106          TI = 0; /* clear interrupt request flag */
107          if (ostart != oend) { /* if characters in buffer and */
108              if (!sendstop) { /* if not Control+S received */
109                  SBUF = outbuf[ostart++ & (OLEN-1)]; /* transmit character */
110                  sendfull = 0; /* clear 'sendfull' flag */
111                  /* if task waiting: signal ready */
112                  if (otask != 0xFF) isr_send_signal (otask);
113              }
114          }
115          else sendactive = 0; /* if all transmitted clear 'sendactive'*/
116      }
117  }
```

V novom ovládači pre sériové rozhranie RS232 boli použité nové užívateľom napísané funkcie **putchar()** a **_get_key()** ktoré nahrádzajú implicitné funkcie z knižnice.

15.17. Univerzálny ovládač komunikačného I²C rozhrania

c:\Omega\data\Dropbox\Záloha\C51\I2C by MAXIM in HEX\I2C_Driver.C

```
1  #include <reg552.h>
2
3  #define Dlzka 6
4
5  extern void Delay (unsigned char);
6
7  void I2C_Start (void)
8  {
9      SCL=SDA=1;    //AGSI drivers vyzaduje !!!, Zacalo davat dobre vysledky s AGSI drivers
10     Delay (Dlzka); //AGSI drivers vyzaduje !!!, Zacalo davat dobre vysledky s AGSI drivers
11     SDA=0;
12     Delay (Dlzka);
13     SCL=0;
14     Delay (Dlzka);
15 }
16
17 void I2C_Bit (unsigned char Data)
18 {
19     SDA=Data; Delay (Dlzka); SCL=1; Delay (Dlzka); SCL=0; Delay (Dlzka);
20 }
21
22 unsigned char I2C_ReadBit (void)
23 {
24     unsigned char Result;
25     #ifndef SIMUL
26         SDA=1; //Zacalo davat dobre vysledky ...
27     #endif
28     Delay (Dlzka);
29     SCL=1;
30     Delay (Dlzka);
31     Result =SDA;
32     Delay (Dlzka);
33     SCL=0;
34     Delay (Dlzka);
35     return (Result);
36 }
37
38 void I2C_Stop (void)
39 {
40     SDA=0; Delay (Dlzka); SCL=1; Delay (Dlzka); SDA=1; Delay (Dlzka);
41 }
42
43 unsigned char I2C_Read (unsigned char doACK)
44 {
45     unsigned char i, Result =0;
46     for (i=0x00; i<0x08; i++)
47     {
48         Result <<= 1;
49         Result |= (I2C_ReadBit () & 0x01);
50     }
51     if (doACK == 0) I2C_Bit (1); /* No ACK */ else I2C_Bit (0); /* Do the ACK */
52     return (Result);
53 }
54
55 unsigned char I2C_SendByte (unsigned char Data)
56 {
57     unsigned char i;
58     for (i=0x00; i<0x08; i++)
59     {
60         if (Data & 0x80) I2C_Bit (1); else I2C_Bit (0);
61         Data <<= 1;
62     }
63     return (I2C_ReadBit ());
64 }
65
66 unsigned char I2C_SendAddr (unsigned char address)
67 {
68     I2C_Start ();
69     return (I2C_SendByte (address));
70 }
```

Používané obvody¹⁰⁰ pre meranie teploty je možné nastaviť na 9 až 13¹⁰¹ bitový prevod teploty.

¹⁰⁰ DS1631 je obvodovo a protokolovo čiastočne rovnaký ako DS1624 a DS1621. Obvod DS1631 má príkaz spustenia prevodu teploty 0x51 a u DS1621 – DS1624 je 0xEE.

¹⁰¹ Len pri obvode DS1624. Tento obvod je možné simulovať v prostredí µVision4 s AGSI ako DS1621.

15.18. Prístup do pamäti E²PROM 256Byte v DS1624 cez I²C rozhranie.

c:\Omega\data\Dropbox\Záloha\C51\I2C by MAXIM in HEX\DS1631.c

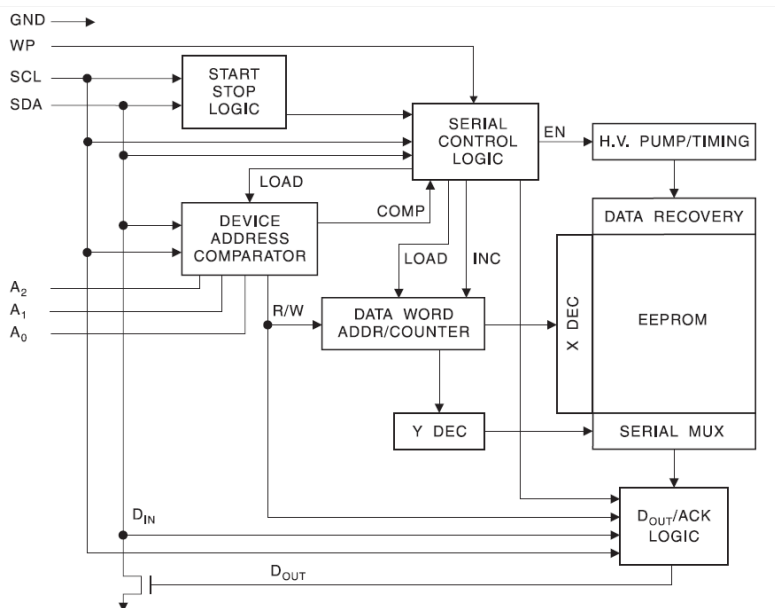
```
172 void Write_Eprom_DS1624 (unsigned char Adresa, unsigned char EEprom_Addr, unsigned char Data)
173 //Funguje zapis a aj citanie do DS1624
174 {
175     I2C_SendAddr (Adresa+WRITE);
176     I2C_SendByte (ACCESS_MEMORY);
177     I2C_SendByte (EEprom_Addr);
178     I2C_SendByte (Data);
179     I2C_Stop ();
180     os_wait (K_TMO,EEPROM_WAIT,NULL); //Cakam 10ms cez RTX51 Full
181 }
182 unsigned char Read_Eprom_DS1624 (unsigned char Adresa, unsigned char EEprom_Addr) //Funguje
183 zapis a aj citanie do DS1624
184 {
185     register unsigned char LSB;
186     I2C_SendAddr (Adresa+WRITE);
187     I2C_SendByte (ACCESS_MEMORY);
188     I2C_SendByte (EEprom_Addr);
189     I2C_SendAddr (Adresa+READ);
190     LSB=I2C_Read (0);
191     I2C_Stop ();
192     return (LSB);
193 }
194 void Write_Block_E2prom_DS1624 (unsigned char Adresa, unsigned char EEprom_Addr, unsigned char
195 Dlzka, unsigned char *Buffer)
196 {
197     #ifndef RTX51
198     register unsigned char Counter=60;
199     #endif
200     register unsigned char i;
201     for (i=0x00; i<Dlzka; i++)
202     {
203         Write_Eprom_DS1624 (Adresa,EEprom_Addr+i,Buffer[i]); //Funguje zapis a aj citanie do DS1624
204     }
205 }
206 void Read_Block_E2prom_DS1624 (unsigned char Adresa, unsigned char EEprom_Addr, unsigned char
207 Dlzka, unsigned char *Buffer)
208 {
209     register unsigned char i;
210     for (i=0x00; i<Dlzka; i++)
211     {
212         Buffer[i]=Read_Eprom_DS1624 (Adresa,EEprom_Addr+i); //Funguje zapis a aj citanie do DS1624
213     }
214 }
```

Pri zápise do E²PROM¹⁰² je treba mať na pamäti, že je nutné počkať cca. 10ms pri 20°C (50ms max.) na ukončenie zápisového cyklu do pamäti E²PROM. Až prijatí I2C_Stop() sa skutočne uskutoční zápis z internej pamäti RAM obvodu DS1624 do E²PROM.

¹⁰² Pamäť je možné použiť aj v jednoduchom i sekvenčnom zápisovom cykle.

15.19. Pamäť E²PROM s rozhraním I²C

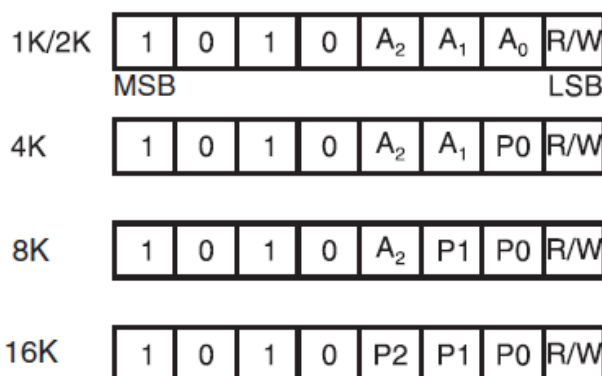
15.19.1. Bloková organizácia pamäti AT24C32/64.



Charakteristické vlastnosti pamäte AT24C32.

- vyrobená CMOS technológiou,
- organizácia¹⁰³ 4096 slov s 8 bitmi,
- obojsmerné I²C rozhranie,
- napájacie napätie od 1,8V do 5,5V,
- 10⁶ zápisových cyklov, doba uloženia 100 rokov.

Obrázok 123, Tvorba adresy E²PROM pamäti AT24C01 až AT24C16



¹⁰³ Pre AT24C01 až AT24C16 je potrebné odoslať len 8 bitovú adresu cez I²C v režime čítania a zápisu. Ovládač je zhodný pre E²PROM pamäti radu AT24C01 až AT24C64, prípadne po malých úpravách blokového zápisu na 64Byte aj pamäť 24C256. Adresa v tomto prípade je tvorená kombináciou adresy obvodu a výberových bitov P0, P1 a P2, ktoré predstavujú tzv. stránku pamäti. Pamäť M24256 nie je odporúčaná pre nové aplikácie. Technické informácie o pamäti E²PROM 24C256 sú <http://www.techdesign.be/projects/datasheet/24LC256.pdf>. AT24C32 a vyššie už vyžadujú 16 bitovú adresu na zbernici pri zápise a čítaní z/do pamäti.

15.19.2. Funkcie jednoduchého zápisu a čítania pre AT24C01 až AT24C16 E²PROM

c:\Omega\data\Dropbox\Záloha\C51\I2C by MAXIM in HEX\AT24Cxx.c

```
1  #include <reg552.h>
2  #include <rtx51.h>
3  #include <stdio.h>
4  #include <stdlib.h>
5  #include <dsw.h>
6
7  #define READ 1
8  #define WRITE 0
9
10 #define EEPROM_WAIT 1
11
12 extern void I2C_Start(void);
13 extern void I2C_Stop(void);
14 extern void Delay(unsigned char Dlzka);
15 extern void I2C_SendAddr(unsigned char Adresa);
16 extern void I2C_SendByte(unsigned char Data);
17 extern unsigned char I2C_Read(unsigned char);
18
19 void Write_Eprom_AT24Cxx(unsigned char Adresa, unsigned int EEprom_Addr, unsigned char Data)
20 {
21     unsigned char subaddr;
22     subaddr=(high(EEprom_Addr))*2; //Pre pamati AT24C01 az AT24C16
23     I2C_SendAddr(Adresa+subaddr+WRITE);
24     I2C_SendByte(low(EEprom_Addr));
25     I2C_SendByte(Data);
26     I2C_Stop();
27     os_wait(K_TMO,EEPROM_WAIT,NULL); //Cakam 10ms cez RTX51 Full
28 }
29
30 unsigned char Read_Eprom_AT24Cxx(unsigned char Adresa, unsigned int EEprom_Addr)
31 {
32     unsigned char LSB,subaddr;
33     subaddr=(high(EEprom_Addr))*2; //Pre pamati AT24C01 az AT24C16
34     I2C_SendAddr(Adresa+subaddr+WRITE);
35     I2C_SendByte(low(EEprom_Addr));
36     I2C_SendAddr(Adresa+READ);
37     LSB=I2C_Read(0);
38     I2C_Stop();
39     return (LSB);
40 }
```

15.19.3. Funkcie jednoduchého zápisu a čítania pre AT24C32 4kB E²PROM

c:\Omega\data\Dropbox\Záloha\C51\I2C by MAXIM in HEX\AT24C32.c

```
19 void Write_Eprom_AT24C32 (unsigned char Adresa, unsigned int EEprom_Addr, unsigned char Data)
20 {
21     I2C_SendAddr (Adresa+WRITE);
22     I2C_SendByte (high (EEprom_Addr)); //Pre AT24C08 tento riadok vymazať, je to 1kB pamat
23     I2C_SendByte (low (EEprom_Addr));
24     I2C_SendByte (Data);
25     I2C_Stop ();
26     os_wait (K_TMO,EEPROM_WAIT,NULL); //Cakam 10ms cez RTX51 Full
27 }
28
29 unsigned char Read_Eprom_AT24C32 (unsigned char Adresa, unsigned int EEprom_Addr)
30 {
31     unsigned char LSB;
32     I2C_SendAddr (Adresa+WRITE);
33     I2C_SendByte (high (EEprom_Addr)); //Pre AT24C08 tento riadok vymazať, je to 1kB pamat
34     I2C_SendByte (low (EEprom_Addr));
35     I2C_SendAddr (Adresa+READ);
36     LSB=I2C_Read (0);
37     I2C_Stop ();
38     return (LSB);
39 }
```

15.19.4. Funkcie blokového zápisu a čítania pre AT24C32 4kB E²PROM

c:\Omega\data\Dropbox\Záloha\C51\I2C by MAXIM in HEX\AT24C32.c

```
41 #define SizeOfEpromBlock 32 //Velkost stranky 32Byte u AT24C32 pre blokovy
    prenos
42
43 void Write_Block_Eprom_AT24C32 (unsigned char Adresa, unsigned char EEprom_Page, unsigned char *
    Buffer)
44 {
45     #ifndef RTX51
46         unsigned char Counter=60;
47     #endif
48     unsigned char i;
49     I2C_SendAddr (Adresa+WRITE);
50     I2C_SendByte (high (EEprom_Page*SizeOfEpromBlock)); //Pre AT24C08 tento riadok vymazať,
    je to 1kB pamat
51     I2C_SendByte (low (EEprom_Page*SizeOfEpromBlock));
52     for (i=0x00; i<SizeOfEpromBlock; i++)
53     {
54         I2C_SendByte (Buffer[i]);
55         os_wait (K_TMO,EEPROM_WAIT,NULL); //Cakam 10ms cez RTX51 Full
56     }
57     I2C_Stop ();
58 }
59
60 void Read_Block_Eprom_AT24C32 (unsigned char Adresa, unsigned char EEprom_Page, unsigned char *
    Buffer)
61 {
62     unsigned char i;
63     I2C_SendAddr (Adresa+WRITE);
64     I2C_SendByte (high (EEprom_Page*SizeOfEpromBlock)); //Pre AT24C08 vymazať, je to 1kB pamat
65     I2C_SendByte (low (EEprom_Page*SizeOfEpromBlock));
66     I2C_SendAddr (Adresa+READ);
67     for (i=0x00; i<SizeOfEpromBlock; i++) //Prenasam 32Byte blok udajov s pACK
68     {
69         switch (i)
70         {
71             case (SizeOfEpromBlock-1) : Buffer[i]=I2C_Read (0); break; // nACK pre ukoncenie
72             default : Buffer[i]=I2C_Read (1); break; //pACK pre udrziavanie
73         }
74     }
75     I2C_Stop ();
76 }
```

15.20. AD prevodník v 80C552

15.20.1. Ovládač pre 10 bitový A/D prevodník v 80C552 v assembleri

C:\Omega\Data\Dropbox\Záloha\C51\AD in C\ADC Converter.asm

```
1 ;-----
2 Repeat macro Num
3     anl a, #Num
4     rl a
5     rl a
6     endm
7 ;-----
8 public _ADC_CONVERSION ;ADC_Conversion(unsigned char Channel)
9 public ?PR?CONVERSION ;Conversion(unsigned char Channel)
10 ;-----
11 ADCON data 0C5h
12 ADCH data 0C6h
13 ;-----
14 ADCS bit acc.3 ;Spustim prevod AD prevodnika
15 ADCI bit acc.4 ;Kontrolujem ci je prevod ukonceny ADCON.4...
16 MASKA equ 11000000B ;D[6..7] ADCH alebo ADCON[6..7]
17 ;-----
18 Prog_ADC_Converter segment code
19 rseg Prog_ADC_Converter
20 ;-----
21 _ADC_CONVERSION: mov a,r7 ;Vstupny parameter je cislo kanala pre meranie v R7
22 setb ADCS ;Nastavim "ADC prevod"
23 xch a,ADCON ;Obsah A potrebujem do ADCON aby sa AD prevod
;spustil
24 ?PR?ADC_END: mov a,ADCON ;ADCI signalizuje ukonceny prevod a pritomnost
;vysledku
25 jnb ADCI,?PR?ADC_END ;Je prevod ukonceny ???
26 clr ADCI ;bit ADCI nastavujem na 0
27 mov ADCON,a
28 ?PR?CONVERSION: Repeat MASKA ;LSB vysledku je v D7,D6 v ADCON[6..7]
29 mov r7,a ;LSB AD prevodu 2 bitov R7=(ADCON[6..7]<<2) ...
30 mov a,ADCH
31 Repeat ~MASKA ;Ziskavam ADCH[0..5]<<2 & (~MASKA) a scitam ich s R7
32 orl a,r7 ;Scitavam ADCH[2..7] | R7
33 xch a,r7 ;vysledok ADCH[2..7] | (ADCON[6..7]<<2)
34 mov a,ADCH
35 Repeat MASKA ;Ziskavam ADCH[6..7]&MASKA
36 mov r6,a ;Vysledok je MSB AD prevodu
37 ret
38 ;-----
39 ; cseg at 0053h ;os_attach_interrupt(10)
40 ; push acc
41 ; pop acc
42 ; reti
43 ;-----
44 end
```

Vykonávanie tejto rutiny trvá¹⁰⁴ 55µs pri 18.432MHz oscilátore. Rovnaký AD prevodník obsahuje aj mikroprocesor DS87C550 od firmy Maxim Integrated.¹⁰⁵ Prevod v tomto prípade je 16 periód¹⁰⁶ oscilátora.

¹⁰⁴ S volaním C a odovzdaním parametrov trvá rutina celkovo 63µs.

¹⁰⁵ <http://www.maximintegrated.com/>, firma ukončila oficiálne jeho výrobu. Pinovo a obvodo je kompatibilný s mikroprocesorom Philips 80C552. DS87C550 obsahuje navyše ešte port P6 ktorý obsadzuje nepoužívané vývody obvodu PLCC.

¹⁰⁶ Podrobný vzťah na výpočet prevodu je uvedený na <http://pdfserv.maximintegrated.com/en/ds/DS87C550.pdf>.

15.20.2. Ovládač pre 10 bitový A/D prevodník v 80C552 v jazyku C

C:\Omega\Data\Dropbox\Záloha\C51\AD in C\AD in C.c

```
1  #include <reg552.h>
2  #include <rtx51.h>
3  #include <stdlib.h>
4  #include <stdio.h>
5  #include <intrins.h>
6
7  #define ADCS 0x08
8  #define ADCI 0x10
9  #define ADEX 0x20
10
11 extern int ADC_Conversion(unsigned char);
12
13 int Voltage;
14
15 int ADC(unsigned char Channel) //Rutina trva 55us pri 18.432MHz
16 {
17     register unsigned int x;
18     ADCON=Channel; //Vyber kanalu pre prevod 0..7
19     ADCON|=(ADEX+ADCS); //Spustim prevod, bolo tu ADCON|=ADEX+ADCS;
20     while((ADCON&ADCI)==0x00); //Cakam na ukoncenie prevodu
21     ADCON^=ADCI; //Nulujem priznak AD prevodnika
22     x=ADCH*0x04; //Normovanie na (int)
23     switch(ADCON&(0x80+0x40))
24     {
25         case 0x40 : x+=0x01; break;
26         case 0x80 : x+=0x02; break;
27         case 0xC0 : x+=0x03; break;
28     }
29     return(x);
30 }
```

Vykonávanie tejto rutiny trvá¹⁰⁷ 55µs pri 18.432MHz oscilátore.

```
31 unsigned int ADCC(unsigned char Channel) //Rutina trva 97us pri 18.432MHz aj s volanim C
32 {
33     ADCON=Channel; //Vyber kanalu pre prevod 0..7
34     ADCON|=ADCS; //Spustim prevod, bolo tu ADCON|=ADEX+ADCS;
35     while((ADCON&ADCI)==0x00); //Cakam na ukoncenie prevodu cca. 80us
36     ADCON^=ADCI; //Nulujem priznak AD prevodnika
37     return(256*ADCH+(ADCON&0xC0)>>6);
38 }
```

Vykonávanie tejto rutiny trvá 97µs pri 18.432MHz oscilátore.

15.21. Použitie a simulácia pamäti E²PROM v µVision4

Mikroprocesory AT89C51ED2 a iné majú integrovanú energeticky nezávislú pamäť E²PROM ktorá sa používa na ukladanie konštánt, alebo premenných použitých v programe. Počet zápisových cyklov do tejto pamäti obmedzený technologicky približne na 10⁵ zápisových cyklov. Počet čítacích cyklov z pamäti E²PROM je neobmedzený. Program µVison4 dokáže pomocou LX51¹⁰⁸, alebo AX51 pristupovať do tejto pamäti¹⁰⁹ ktorá je deklarovaná ako **far** umiestnená v pamäťovom priestore v rozsahu X:0x020000 - X:0x0207FF.¹¹⁰

¹⁰⁷ S volaním C a odovzdaním parametrov trvá rutina celkovo 67µs.

¹⁰⁸ S operačným systémom RTX51 Tiny alebo RTX51 Full po naprogramovaní nepracuje správne a nie je možné kvôli neustálym resetom odladiť napísaný program !!!

¹⁰⁹ Pre správnu činnosť je potrebné zapnúť mód LX51, AX51 prekladača. Pre možnosti simulácie viď. stránku http://www.keil.com/uvision/db_sim_prf_flash.asp

¹¹⁰ Viď. DS89C390, alebo DS80C400, DS80C410, DS80C411.

Obrázok 124, Použitie E²PROM pamäti v programe

```
1  /*
2  * This file contains the data definitions that are stored in EEPROM
3  */
4
5  #ifndef DECLARE // if DECLARE is not defined
6  #define DECLARE extern // variables are extern
7  #endif
8
9  struct sample {
10     char carray[20];
11     int iarray[20];
12     long larray[20];
13 };
14
15 DECLARE struct sample far esample1; // reserve space for esample1 struct
16 DECLARE struct sample far esample2; // reserve space for esample2 struct
17 DECLARE char * far etxt_ptr; // pointer in EEPROM space
```

15.22. Elektronický potenciometer DS1807 s I²C

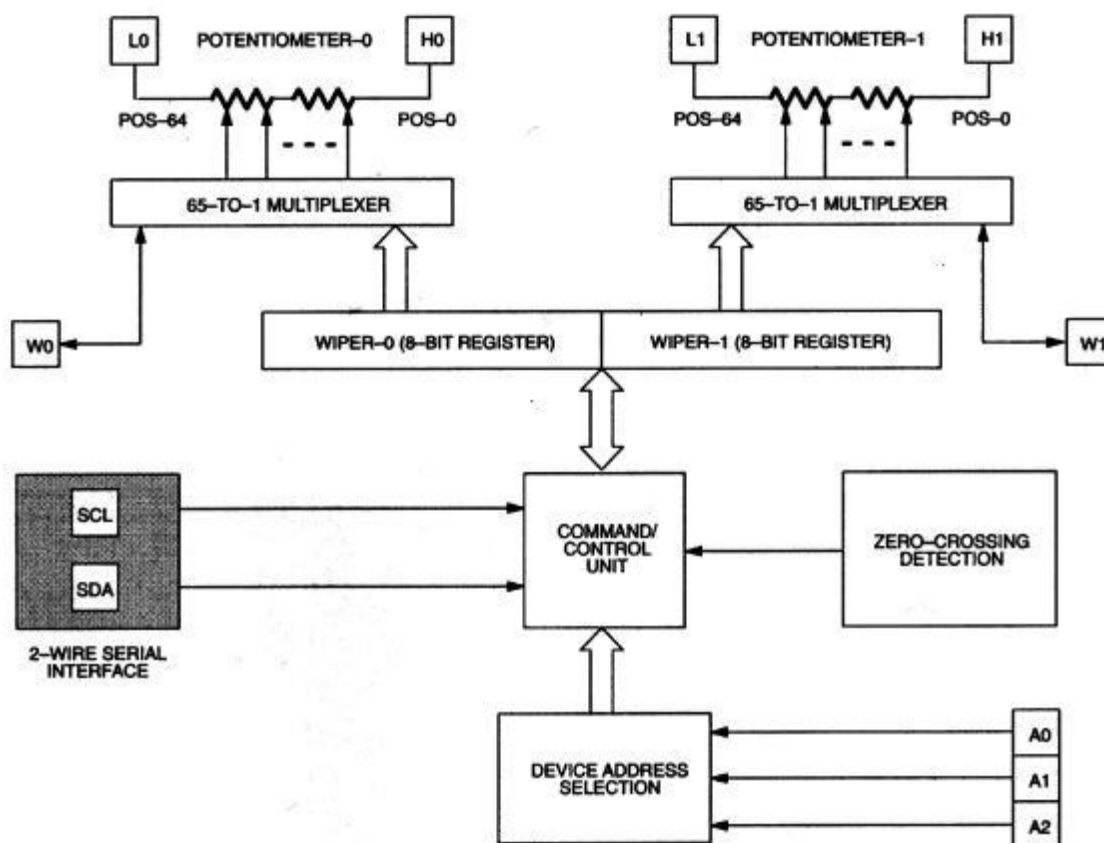
```
1  #include <reg552.h>
2  #include <rtx51.h>
3  #include <stdio.h>
4  #include <stdlib.h>
5  #include <dsw.h>
6
7  #define I2C_READ 1
8  #define I2C_WRITE 0
9
10 #define EEPROM_WAIT 1
11
12 extern void Delay(unsigned char Dlzka);
13 extern void I2C_SendAddr(unsigned char Data, unsigned char ACK);
14 extern void I2C_SendByte(unsigned char Data);
15 extern unsigned char I2C_Read(unsigned char);
16 extern void I2C_Start(void);
17 extern void I2C_Stop(void);
18
19 #define DS1807 0x50 //CONTROL BYTE t.j. Idendifikacne cislo - adresa DS1807
20 #define POT0 0xA9 //COMMAND
21 #define POT1 0xAA //COMMAND
22 #define POTALL 0xAF //COMMAND
23
24 void Write_DS1807(unsigned char Adresa, unsigned int Data)
25 {
26     #ifndef RTX51
27         register unsigned char Counter=60;
28     #endif
29     I2C_SendAddr(Adresa, I2C_WRITE);
30     I2C_SendByte(POT0);
31     I2C_SendByte(high(Data)); //Data pre POT1
32     I2C_SendByte(low(Data)); //Data pre POT0
33     I2C_Stop();
34     os_wait(K_TMO, EEPROM_WAIT, NULL);
35 }
36
37 unsigned int Read_DS1807(unsigned char Adresa)
38 {
39     register unsigned char LSB, MSB;
40     I2C_SendAddr(Adresa, I2C_READ);
41     MSB=I2C_Read(1); //Data pre POT1
42     LSB=I2C_Read(0); //Data pre POT0
43     I2C_Stop();
44     return (MSB*0x100+LSB);
45 }
```

Elektronický duálny logaritmický odporový potenciometer¹¹¹ DS1807 sa používa v spotrebnej elektronike nielen na reguláciu hlasitosti, ale aj ako elektronický potenciometer ovládaný po rozhraní I²C. Jeden krok v tomto prípade tu predstavuje 1dB a 64 krok až 90dB útlm¹¹². Súčasne môže pracovať na rovnakej I²C zbernici až 8 obvodov.

¹¹¹ 6-bitový.

¹¹² Funkcia MUTE.

Obrázok 125, Vnútorná schéma DS1807



15.23. Funkcie I²C pre DS1631 v jazyku C51

c:\Omega\data\Dropbox\Záloha\C51\I2C by MAXIM in HEX\DS1631.c

```
1  #include <rtx51.h>
2  #include <reg552.h>
3  #include <stdio.h>
4  #include <stdlib.h>
5  #include <dsw.h>
6
7  #define K (1.0/256.0)
8
9  #ifndef READ
10     #define READ 1
11 #endif
12 #ifndef WRITE
13     #define WRITE 0
14 #endif
15
16 #define EEPROM_WAIT 1
17
18 extern void Delay (unsigned char);
19
20 extern void I2C_Start (void);
21 extern void I2C_Stop (void);
22 extern unsigned char I2C_Read (unsigned char);
23 extern unsigned char I2C_SendByte (unsigned char);
24 extern unsigned char I2C_SendAddr (unsigned char);
25 extern void Write_TH_Register (unsigned char, unsigned int);
26 extern void Write_TL_Register (unsigned char, unsigned int);
27 extern unsigned int Read_TH (unsigned char);
28 extern unsigned int Read_TL (unsigned char);
29
30 #ifdef DS1631
31     #define START_CONVERT 0x51 //DS1631 je maximálne 12bitový teplomer
32     //DS1631 I2C_START_CONVERT = 0x51, DS1624 I2C_START_CONVERT
33     = 0xEE
34 #endif
35
36 #ifdef DS1624
37     #define START_CONVERT 0xEE //DS1624 je maximálne 13bitový teplomer
38     //DS1631 I2C_START_CONVERT = 0x51, DS1624 I2C_START_CONVERT
39     = 0xEE
40 #endif
41
42 #define ACCESS_MEMORY 0x17 //DS1624
43 #endif
44
45 #define STOP_CONVERT 0x22
46 #define ACCESS_TH 0xA1
47 #define ACCESS_TL 0xA2
48 #define READ_TEMPERATURE 0xAA
49 #define DS1631_COMMAND 0xAC
50 #define DS1624_COMMAND 0xAC
51 #define COMMAND_DONE 0x80
52 #define COMMAND_1SHOT 0x01
53 #define POR 0x54
54 #define READ_COUNTER 0xA8 //DS1621
55 #define READ_SLOPE 0xA9 //DS1621
56
57 #define AddrDS1631A 0x90 //Adresa pre DS1621,DS1624,DS1631 - tzv. CONTROL_BYTE
58 #define AddrDS1624A 0x90 //Adresa pre DS1621,DS1624,DS1631 - tzv. CONTROL_BYTE
59 #define AddrAT24C32 0xA0 //Adresa pre AT24C08 az AT24C64 - tzv. CONTROL_BYTE
60 #define AddrDS1807A 0x50 //Adresa pre DS1807A - tzv. CONTROL_BYTE
61
62 unsigned char Get_Config (unsigned char Adresa)
63 {
64     register unsigned char LSB;
65     I2C_SendAddr (Adresa+WRITE);
66     I2C_SendByte (DS1631_COMMAND);
67     I2C_SendAddr (Adresa+READ);
68     LSB=I2C_Read (1);
69     I2C_Stop ();
70     return (LSB);
71 }
72
73 void Write_Config (unsigned char Adresa, unsigned char Data)
74 {
75     I2C_SendAddr (Adresa+WRITE);
76     I2C_SendByte (DS1631_COMMAND);
77     I2C_SendByte (Data);
78     os_wait (K_TMO,EEPROM_WAIT,NULL); //Cakam 10ms cez RTX51 Full
79 }
```

Page1

c:\Omega\data\Dropbox\Záloha\C51\I2C by MAXIM in HEX\DS1631.c

```
73     I2C_Stop ();
74 }
75
76 void Config_DS16xx (unsigned char Adresa, unsigned char Bits)
77 {
78     register unsigned char Config;
79     register unsigned int Temp;
80     Temp = Read_TH (AddrDS1631A);
81     Temp = Read_TL (AddrDS1631A);
82     Write_TH_Register (AddrDS1631A, 0xabcd);
83     Write_TL_Register (AddrDS1631A, 0x1234);
84     Config = Get_Config (Adresa);
85     Config &= ~(0x08 + 0x04);
86     switch (Bits)
87     {
88         case 9 : break;
89         case 10 : Config |= 0x04; break;
90         case 11 : Config |= 0x08; break;
91         default : Config |= 0x0c; break; //Ak je zadana zla hodnota alebo 12 tak nastavim 12 bitov
92     }
93     Write_Config (Adresa, Config); //Nastavim 12 bitovy prevod
94 }
95
96 unsigned int Get_Int_Temp (unsigned char Adresa)
97 {
98     register unsigned char MSB, LSB;
99     I2C_SendAddr (Adresa + WRITE);
100    I2C_SendByte (START_CONVERT);
101    I2C_Stop ();
102    //-----
103    I2C_SendAddr (Adresa + WRITE);
104    I2C_SendByte (STOP_CONVERT);
105    I2C_Stop ();
106    //-----
107    I2C_SendAddr (Adresa + WRITE);
108    I2C_SendByte (READ_TEMPERATURE);
109    I2C_SendAddr (Adresa + READ);
110    MSB = I2C_Read (1); //Vyzaduje sa zo SLAVE-a potvrdenie ACK=0!!!
111    LSB = I2C_Read (0); //Vyzaduje sa z MASTER-a potvrdenie ACK=1!!!
112    I2C_Stop ();
113    //*****
114    if (MSB >= 0x80) return ((MSB * 256 + LSB) - 65536); else return (MSB * 256 + LSB);
115    //Je to OK !!!
116    //*****
117 }
118
119 float Get_Float_Temp (unsigned char Adresa)
120 {
121     register unsigned char MSB, LSB;
122     I2C_SendAddr (Adresa + WRITE);
123     I2C_SendByte (START_CONVERT);
124     I2C_Stop ();
125     //-----
126     I2C_SendAddr (Adresa + WRITE);
127     I2C_SendByte (STOP_CONVERT);
128     I2C_Stop ();
129     //-----
130     I2C_SendAddr (Adresa + WRITE);
131     I2C_SendByte (READ_TEMPERATURE);
132     I2C_SendAddr (Adresa + READ);
133     MSB = I2C_Read (1); //Vyzaduje sa zo SLAVE-a potvrdenie ACK=0!!!
134     LSB = I2C_Read (0); //Vyzaduje sa z MASTER-a potvrdenie ACK=1!!!
135     I2C_Stop (); //MSB=0x19; LSB=0x10; //25.062°C
136     //*****
137     if (MSB >= 0x80) return ((float) (MSB * 256 + LSB) - 65536); else return ((float) (MSB * 256 + LSB) * K); //Je
138     to OK !!!
139     //*****
140 }
141
142 void Write_TH_Register (unsigned char Adresa, unsigned int THReg)
143 {
144     I2C_SendAddr (Adresa + WRITE);
145     I2C_SendByte (ACCESS_TH); // !!!!!!!!!!!!!!!!!!!!!!!
146     I2C_SendByte (high (THReg));
```

Page2

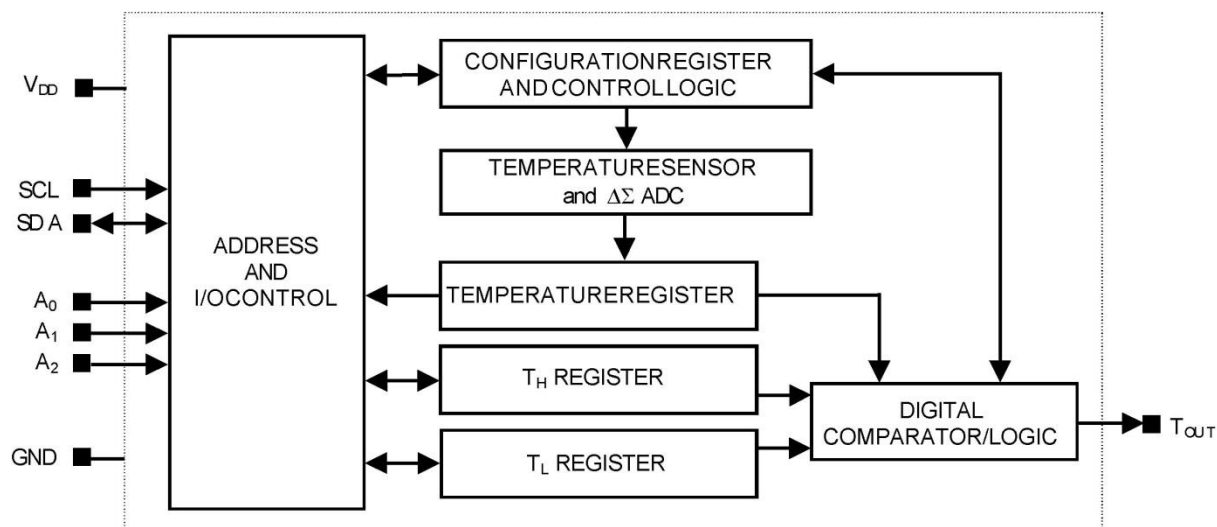
c:\Omega\data\Dropbox\Záloha\C51\I2C by MAXIM in HEX\DS1631.c

```
145 I2C_SendByte (low (THReg));
146 I2C_Stop ();
147 os_wait (K_TMO,EEPROM_WAIT,NULL); //Cakam 10ms cez RTX51 Full
148 }
149
150 void Write_TL_Register (unsigned char Adresa, unsigned int TLReg)
151 {
152 I2C_SendAddr (Adresa+WRITE);
153 I2C_SendByte (ACCESS_TL); // !!!!!!!!!!!!!!!!!!!!!!!
154 I2C_SendByte (high (TLReg));
155 I2C_SendByte (low (TLReg));
156 I2C_Stop ();
157 os_wait (K_TMO,EEPROM_WAIT,NULL); //Cakam 10ms cez RTX51 Full
158 }
159
160 unsigned int Read_TH (unsigned char Adresa)
161 {
162 register unsigned char MSB,LSB;
163 I2C_SendAddr (Adresa+WRITE);
164 I2C_SendByte (ACCESS_TH);
165 I2C_SendAddr (Adresa+READ);
166 MSB=I2C_Read (1);
167 LSB=I2C_Read (0);
168 I2C_Stop ();
169 return (MSB*0x100+LSB);
170 }
171
172 unsigned int Read_TL (unsigned char Adresa)
173 {
174 register unsigned char MSB,LSB;
175 I2C_SendAddr (Adresa+WRITE);
176 I2C_SendByte (ACCESS_TL);
177 I2C_SendAddr (Adresa+READ);
178 MSB=I2C_Read (1);
179 LSB=I2C_Read (0);
180 I2C_Stop ();
181 return (MSB*0x100+LSB);
182 }
183
184 #ifdef DS1624
185
186 void Write_Eprom_DS1624 (unsigned char Adresa, unsigned char EEprom_Addr, unsigned char Data)
187 //Funguje zapis a aj citanie do DS1624
188 {
189 I2C_SendAddr (Adresa+WRITE);
190 I2C_SendByte (ACCESS_MEMORY);
191 I2C_SendByte (EEprom_Addr);
192 I2C_SendByte (Data);
193 I2C_Stop ();
194 os_wait (K_TMO,EEPROM_WAIT,NULL); //Cakam 10ms cez RTX51 Full
195 }
196
197 unsigned char Read_Eprom_DS1624 (unsigned char Adresa, unsigned char EEprom_Addr) //Funguje
198 zapis a aj citanie do DS1624
199 {
200 register unsigned char LSB;
201 I2C_SendAddr (Adresa+WRITE);
202 I2C_SendByte (ACCESS_MEMORY);
203 I2C_SendByte (EEprom_Addr);
204 I2C_SendAddr (Adresa+READ);
205 LSB=I2C_Read (0);
206 I2C_Stop ();
207 return (LSB);
208 }
209
210 void Write_Block_E2prom_DS1624 (unsigned char Adresa, unsigned char EEprom_Addr, unsigned char
211 Dlzka, unsigned char *Buffer)
212 {
213 #ifndef RTX51
214 register unsigned char Counter=60;
215 #endif
216 register unsigned char i;
217 for (i=0x00; i<Dlzka; i++)
218 {
```

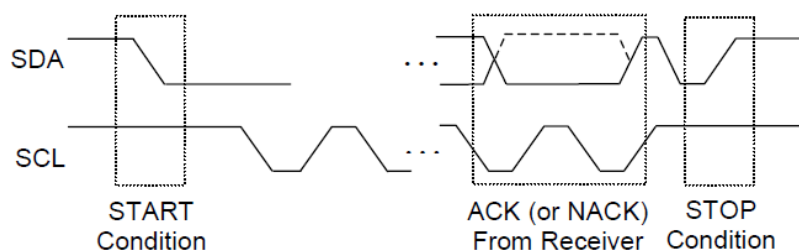

c:\Omega\data\Dropbox\Záloha\C51\I2C by MAXIM in HEX\DS1631.c

```
216 Write_Eprom_DS1624 (Adresa, EEprom_Addr+i, Buffer[i]); //Funguje zapis a aj citanie do DS1624
217 }
218 }
219
220 void Read_Block_E2prom_DS1624 (unsigned char Adresa, unsigned char EEprom_Addr, unsigned char
Dlзка, unsigned char *Buffer)
221 {
222     register unsigned char i;
223     for (i=0x00; i<Dlзка; i++)
224     {
225         Buffer[i]=Read_Eprom_DS1624 (Adresa, EEprom_Addr+i); //Funguje zapis a aj citanie do DS1624
226     }
227 }
228
229 /*
230
231 #define SizeRamBlock      8                //Velkost stranky je 8-Byte u DS1624 pre blokovy prenos
...
232 #define SizeOfEepromBlock 256            //Velkost pamati EEPROM ...
233
234 void Write_Block_Eprom_DS1624(unsigned char Adresa, unsigned char EEprom_Addr, unsigned char
*Buffer)
235 {
236     register unsigned char i, ram_cache=0x01;
237     I2C_SendAddr(Adresa+WRITE);
238     I2C_SendByte(ACCESS_MEMORY);
239     I2C_SendByte(EEprom_Addr);
240     for(i=0x00; i<SizeOfEepromBlock; i++) //Maximalne prenesiem blokovo 8-byte a I2C_Stop()
spusti jeho prepis z RAM do EEPROM !!!
241     {
242         I2C_SendByte(Buffer[i]);
243         if(ram_cache++>SizeRamBlock)
244         {
245             ram_cache=0x01;
246             I2C_Stop();
247             os_wait(K_TMO,EEPROM_WAIT,NULL); //Cakam 10ms cez RTX51 Full
248         }
249     }
250     if((ram_cache!=0x01)&(i!=0x00)) I2C_Stop(); //je to dobre ???
251 }
252
253 void Read_Block_Eprom_DS1624(unsigned char Adresa, unsigned char EEprom_Addr, unsigned char
*Buffer)
254 {
255     register unsigned char i;
256     I2C_SendAddr(Adresa+WRITE);
257     I2C_SendByte(ACCESS_MEMORY);
258     I2C_SendByte(EEprom_Addr);
259     I2C_SendAddr(Adresa+READ);
260     for(i=0x00; i<SizeOfEepromBlock; i++) //Prenasam 256Byte blok
udajov s pACK
261     {
262         switch(i)
263         {
264             case (SizeOfEepromBlock-1) : Buffer[i]=I2C_Read(0); break; // nACK pre ukoncenie
265             default : Buffer[i]=I2C_Read(1); break; // pACK pre udrziavanie
266         }
267     }
268     I2C_Stop();
269 }
270 */
271 #endif
```

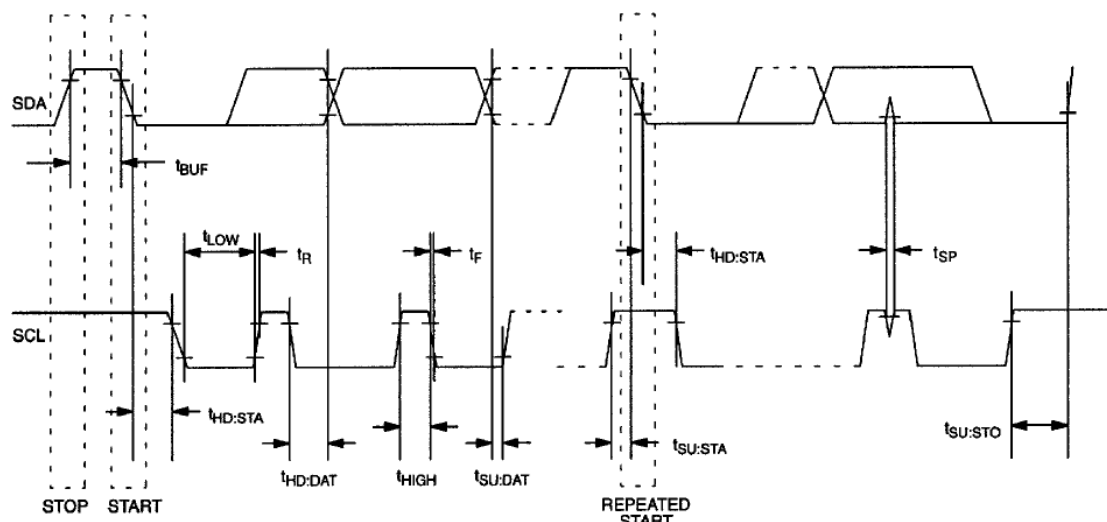
Obrázok 126, Komunikačný model obvodu DS1631 – vnútorné zapojenie¹¹³



Obrázok 127, Symbolické priebehy napätí na I²C rozhraní pri komunikácii

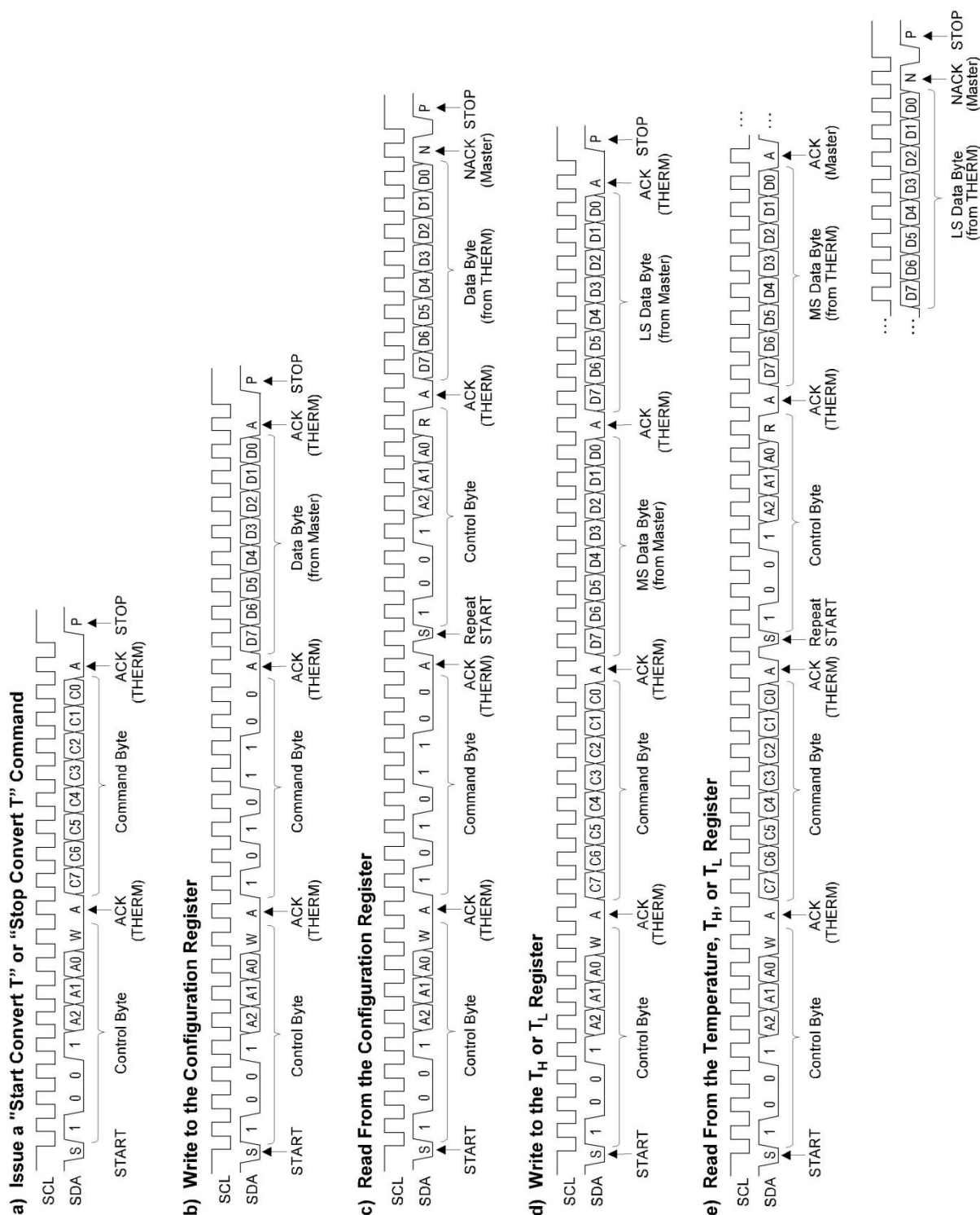


Obrázok 128, Detailné priebehy napätí na I²C rozhraní pri komunikácii



¹¹³ <http://datasheets.maximintegrated.com/en/ds/DS1631-DS1731.pdf>

Obrázok 129, Časové závislosti na I²C rozhraní DS1631



Tabuľka 10, Výstupný formát výsledku prevodu z DS1631

MS Byte	bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8
	S	2^6	2^5	2^4	2^3	2^2	2^1	2^0
LS Byte	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
	2^{-1}	2^{-2}	2^{-3}	2^{-4}	0	0	0	0

Tabuľka 11, Výsledok 12-bitového prevodu z DS1631

TEMPERATURE (°C)	DIGITAL OUTPUT (BINARY)	DIGITAL OUTPUT (HEX)
+125	0111 1101 0000 0000	7D00h
+25.0625	0001 1001 0001 0000	1910h
+10.125	0000 1010 0010 0000	0A20h
+0.5	0000 0000 1000 0000	0080h
0	0000 0000 0000 0000	0000h
-0.5	1111 1111 1000 0000	FF80h
-10.125	1111 0101 1110 0000	F5E0h
-25.0625	1110 0110 1111 0000	E6F0h
-55	1100 1001 0000 0000	C900h

Tabuľka 12, Konfiguračný register DS1631

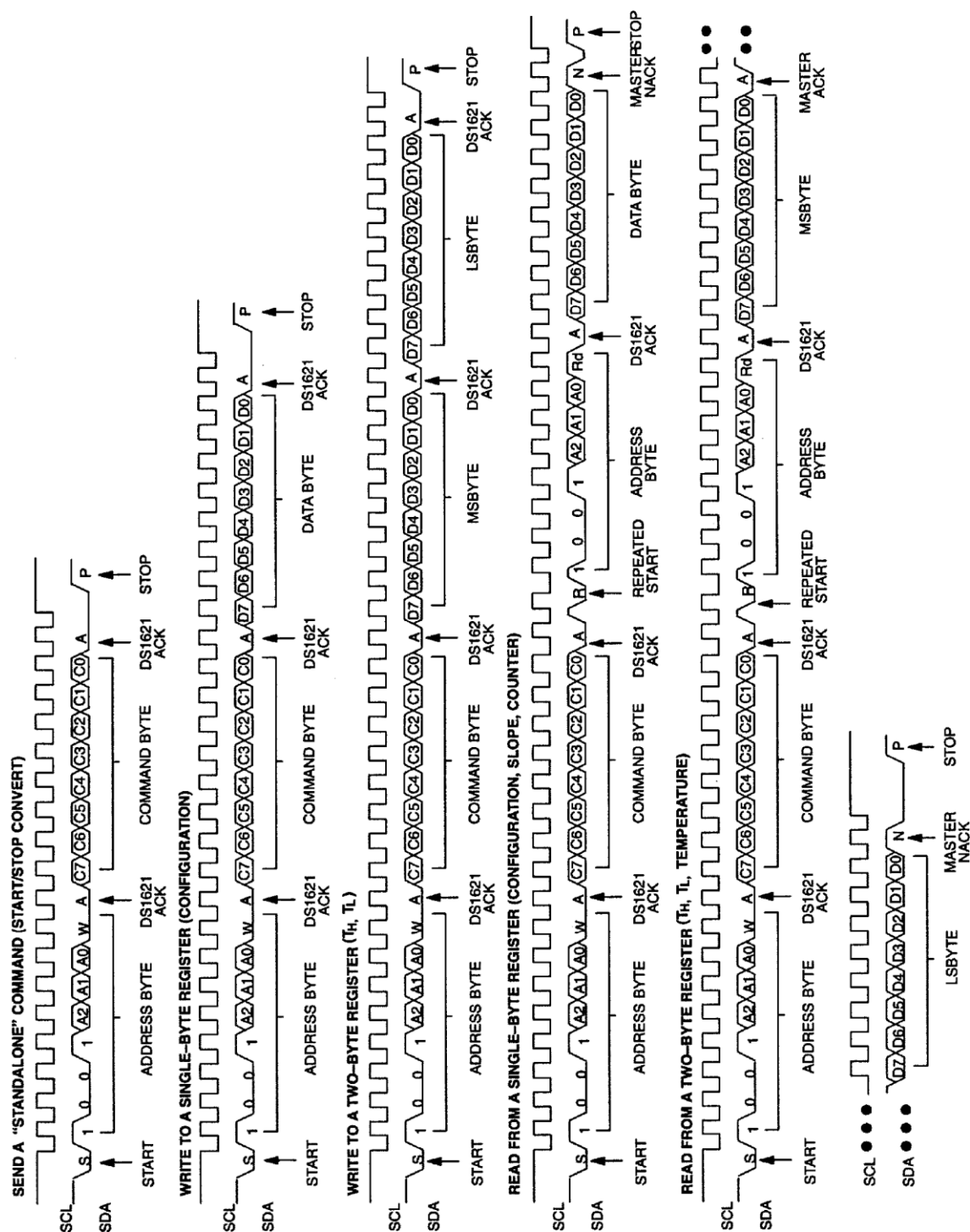
MSb	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	LSb
DONE	THF	TLF	NVB	R1	R0	POL*	1SHOT*

*NV (EEPROM)

Tabuľka 13, Nastavenie presnosti prevodu teploty v DS1631

R1	R0	RESOLUTION (BIT)	CONVERSION TIME (MAX)
0	0	9	93.75ms
0	1	10	187.5ms
1	0	11	375ms
1	1	12	750ms

Obrázok 130, Časové závislosti na I²C rozhraní DS1621



DS1621 je obvodovo a protokolovo zhodný ako DS1624 a je ho možné simulovať v prostredí μ Vision4.

15.24. Zoznam funkcií pre I²C rozhranie

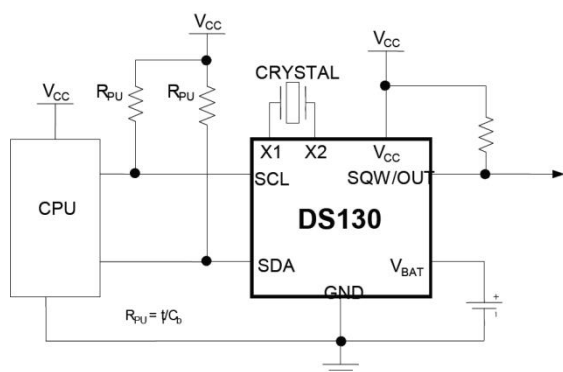
c:\Omega\data\Dropbox\Záloha\C51\I2C by MAXIM in HEX\I2C_by_DSW.c

```
21 //*****
22 extern void Delay (unsigned char);
23 extern void I2C_Start (void);
24 extern void I2C_Stop (void);
25 extern unsigned char I2C_Read (unsigned char doACK);
26 extern unsigned char I2C_SendByte (unsigned char Data);
27 extern unsigned char I2C_SendAddr (unsigned char Addr);
28 #ifdef BLOCK
29 extern unsigned char i2c_writeblock (unsigned char address, unsigned char *barr, int length);
30 extern unsigned char i2c_readblock (unsigned char address, unsigned char *barr, int length);
31 extern unsigned char i2c_writereadblock (unsigned char address, unsigned char *barr1, int length1,
    unsigned char *barr2, int length2);
32 #endif
33 //*****
34 extern unsigned char Get_Config (unsigned char Adresa);
35 void Write_Config (unsigned char Adresa, unsigned char Data);
36 extern void Config_DS16xx (unsigned char Adresa, unsigned char Bits);
37 extern unsigned int Get_Int_Temp (unsigned char Adresa);
38 float Get_Float_Temp (unsigned char Adresa);
39 extern void Write_TH_Register (unsigned char, unsigned int);
40 extern void Write_TL_Register (unsigned char, unsigned int);
41 extern unsigned int Read_TH (unsigned char);
42 extern unsigned int Read_TL (unsigned char);
43 #ifdef DS1624
44 extern void Write_Eprom_DS1624 (unsigned char Adresa, unsigned char EEprom_Addr, unsigned char
    Data); //Pre DS1624
45 extern unsigned char Read_Eprom_DS1624 (unsigned char Adresa, unsigned char EEprom_Addr);
    //Pre DS1624
46 extern void Write_Block_Eprom_DS1624 (unsigned char Adresa, unsigned char EEprom_Addr, unsigned
    char *Buffer);
47 extern void Read_Block_Eprom_DS1624 (unsigned char Adresa, unsigned char EEprom_Addr, unsigned
    char *Buffer);
48 extern void Write_Block_E2prom_DS1624 (unsigned char Adresa, unsigned char EEprom_Addr, unsigned
    char Dlzka, unsigned char *Buffer);
49 extern void Read_Block_E2prom_DS1624 (unsigned char Adresa, unsigned char EEprom_Addr, unsigned
    char Dlzka, unsigned char *Buffer);
50 #endif
51 //*****
52 extern void Write_Eprom_AT24C32 (unsigned char, unsigned int, unsigned char);
53 extern unsigned char Read_Eprom_AT24C32 (unsigned char, unsigned int);
54 extern void Write_Block_Eprom_AT24C32 (unsigned char Adresa, unsigned char EEprom_Addr, unsigned
    char *Buffer);
55 extern void Read_Block_Eprom_AT24C32 (unsigned char Adresa, unsigned char EEprom_Addr, unsigned
    char *Buffer);
56 //*****
57 extern void Write_DS1807 (unsigned char Adresa, unsigned int Data);
58 extern unsigned int Read_DS1807 (unsigned char Adresa);
59 //*****
60 extern void Init_Serial (void);
61 //*****
62 extern int ADC_Conversion (unsigned char);
63 //*****
64
```


15.25. Obvod DS1307 - Real Time Clock

Špecializovaný obvod DS1307 je použiteľný ako hodinový obvod v aplikáciách s mikroprocesormi. Pomocou I²C rozhrania je možné pristupovať k vnútorným registrom obsahujúce čas¹¹⁴, dátum a aj pamäti RAM o veľkosti 56 Byte. Vývod č. 7 v puzdre DIP s názvom SQW/OUT je možné nastaviť na periodické generovanie frekvencie až do 1Hz. Obvod DS1307 k svojej činnosti vyžaduje len kryštál 32.768kHz a je zálohovaný typicky 3V batériou CR2031. Pre svoju správnu funkciu vyžaduje napájacie napätie 5V a v režime zálohy napätie 3V. Odber prúdu v režime na záložnú batériu je typicky 0,5μA a je ho možné minimalizovať vypnutím oscilátora hodín t.j. nastavením bitu CH¹¹⁵ na 1. Údaje sú v tomto režime výrobcom zálohované po dobu minimálne 10 rokov.

Obrázok 131, Schéma zapojenia DS1307+



Tabuľka 14, Vnútorné registre DS1307

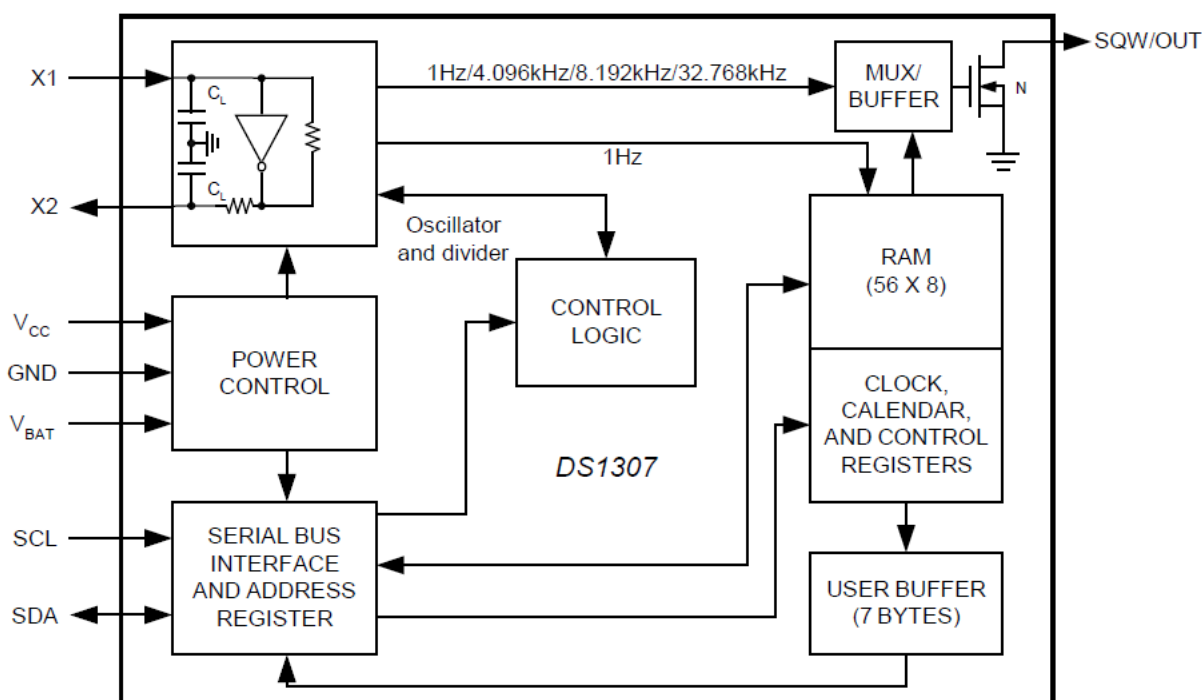
ADDRESS	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0	FUNCTION	RANGE
00h	CH	10 Seconds			Seconds				Seconds	00–59
01h	0	10 Minutes			Minutes				Minutes	00–59
02h	0	12	10 Hour	10 Hour	Hours				Hours	1–12 +AM/PM 00–23
		24	PM/ AM							
03h	0	0	0	0	0	DAY			Day	01–07
04h	0	0	10 Date		Date				Date	01–31
05h	0	0	0	10 Month	Month				Month	01–12
06h	10 Year				Year				Year	00–99
07h	OUT	0	0	SQWE	0	0	RS1	RS0	Control	—
08h–3Fh									RAM 56 x 8	00h–FFh

0 = Always reads back as 0.

¹¹⁴ Vnútorné registre obvodu DS1307 času a dátumu vyžadujú zápis a čítanie údajov tzv. BCD formáte. Zápis do vnútornej pamäti RAM je síce tiež v BCD formáte, ale principiálne je úplne jedno či je číslo zapísané v BCD, alebo v dekadickom tvare.

¹¹⁵ Po prvom pripojení napájacieho napätia je nastavený bit CH=1, takže oscilátor hodín je zastavený!

Obrázok 132, Vnútrotná schéma zapojenia DS1307



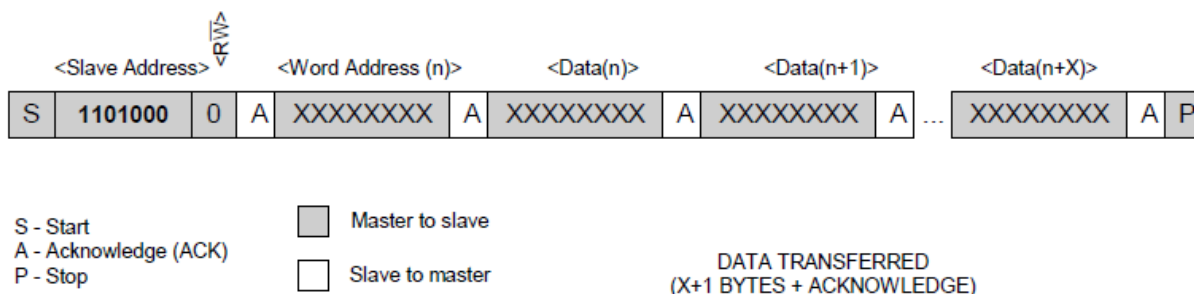
Obrázok 133, Prevod čísla na BCD/DEC formát

c:\Omega\data\Dropbox\Záloha\C51\I2C by MAXIM in HEX\BCD DEC Convert\BCD DEC convert.c

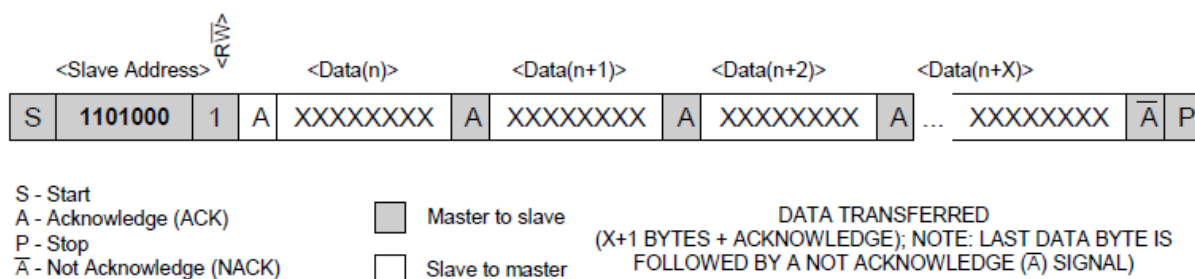
```

1  #include <reg51.h>
2  #include <stdio.h>
3  #include <stdlib.h>
4
5  unsigned char BCD (unsigned char Data )
6  {
7      return ((Data /10 *16)+( Data %10 ));
8  }
9
10 unsigned char DEC (unsigned char Data )
11 {
12     return ((Data /16 *10)+( Data %16 ));
13 }
    
```

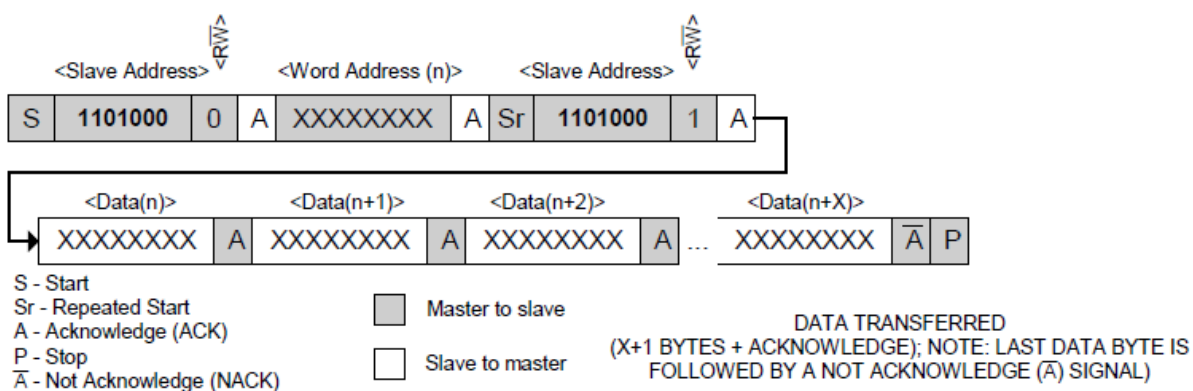
Obrázok 134, Zápis dát v režime Slave Receiver Mode (Blokový mód)



Obrázok 135, Čítanie dát v režime Slave Transmitter Mode (Blokový mód)



Obrázok 136, Kombinované čítanie dát Slave Receive and Transmit (po byte)



Obrázok 137, Komunikačné rutiny pre DS1307 pomocou I²C.

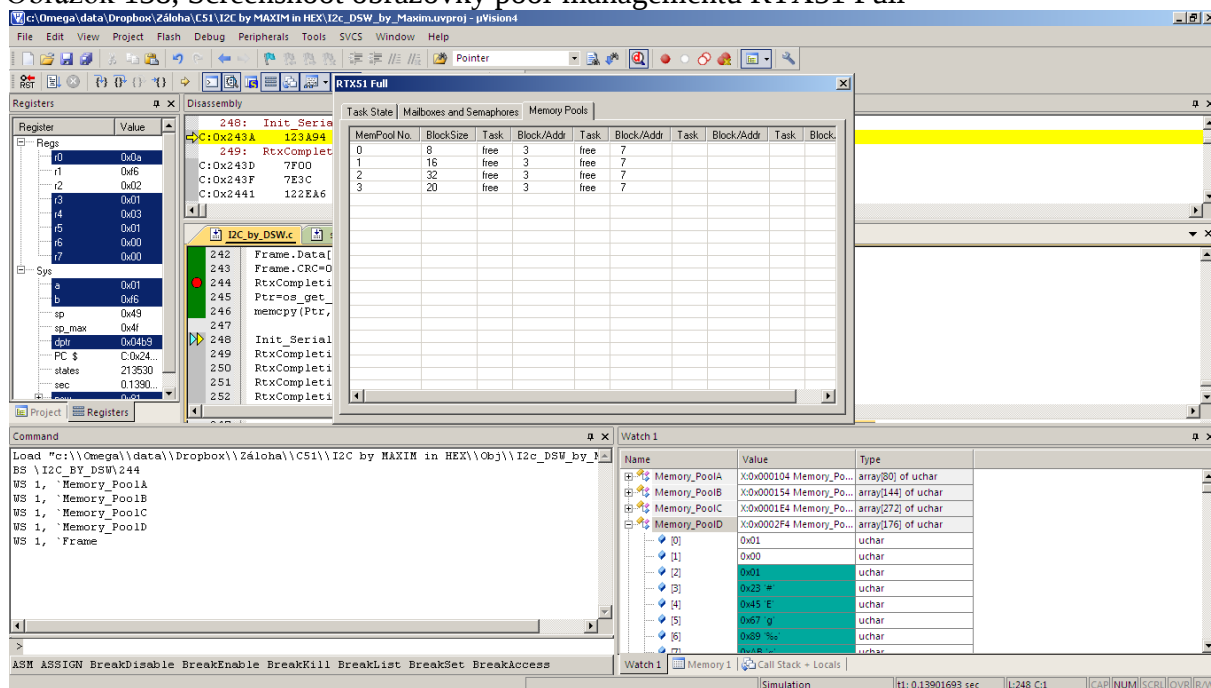
C:\Omega\data\Dropbox\Záloha\C51\DS1307\DS1307+.c

```
1  #include <rtx51.h>
2  #include <reg51.h>
3  #include <stdio.h>
4  #include <stdlib.h>
5  #include <dsw.h>
6  #include <adresy.inc>
7
8  #ifndef READ
9  #define READ 1
10 #endif
11 #ifndef WRITE
12 #define WRITE 0
13 #endif
14
15 #define EEPROM_WAIT 1
16
17 extern void Delay (unsigned char);
18
19 extern void I2C_Start (void);
20 extern void I2C_Stop (void);
21 extern unsigned char I2C_Read (unsigned char);
22 extern unsigned char I2C_SendByte (unsigned char);
23 extern unsigned char I2C_SendAddr (unsigned char);
24
25 #define Radix10 10
26 #define Radix16 16
27
28 unsigned char Get_Time (unsigned char Adresa, unsigned char Addr)
29 {
30     register unsigned char LSB, time;
31     I2C_SendAddr (Adresa+WRITE);
32     I2C_SendByte (Addr);
33     //-----
34     I2C_SendAddr (Adresa+READ);
35     LSB=I2C_Read (0); //1 - Vyzaduje sa zo SLAVE-a potvrdenie ACK=0!!!
36     I2C_Stop (); //0 - Vyzaduje sa z MASTER-a potvrdenie ACK=1!!!
37     //*****
38     time=LSB; //Vraciam vysledok v BCD formate
39     switch (Addr)
40     {
41         case Addr_Sec :
42         case Addr_Min : LSB=((time & 0x70)>>4)*10+(time & 0x0F); break;
43         case Addr_Hod : LSB=((time & 0x30)>>4)*10+(time & 0x0F); break;
44         case Addr_Day : LSB=(time & 0x07); break;
45         case Addr_Den : LSB=((time & 0x30)>>4)*10+(time & 0x0F); break;
46         case Addr_Mes : LSB=((time & 0x10)>>4)*10+(time & 0x0F); break;
47         case Addr_Rok : LSB=((time & 0xF0)>>4)*10+(time & 0x0F); break;
48     }
49     return (LSB); //Vraciam vysledok v DEC formate
50     //*****
51 }
52
53 void Set_Time (unsigned char Adresa, unsigned char Addr, unsigned char Data)
54 {
55     register unsigned char time;
56     time=Data; //Zadavam vysledok v BCD formate
57     switch (Addr)
58     {
59         case Addr_Sec :
60         case Addr_Min : Data=((time /10)&0x07)<<4|(time %10); break;
61         case Addr_Hod : Data=((time /10)&0x03)<<4|(time %10); break;
62         case Addr_Day : Data=(time & 0x07); break;
63         case Addr_Den : Data=((time /10)&0x03)<<4|(time %10); break;
64         case Addr_Mes : Data=((time /10)&0x01)<<4|(time %10); break;
65         case Addr_Rok : Data=((time /10)&0x0F)<<4|(time %10); break;
66     }
67     I2C_SendAddr (Adresa+WRITE);
68     I2C_SendByte (Addr);
69     //-----
70     I2C_SendByte (Data); //Vraciam vysledok v BCD formate
71     I2C_Stop ();
72 }
73
74 /*
```

15.26. Management pamäti v RTX51 Full

RTX51 disponuje výkonným nástrojom pre správu externej pamäti RAM v podobe niekoľkých mocných príkazov uvedených v kapitole 13.7. Nasledujúca časť programu demonštruje ako je možné využiť príkazy správy pamäti v praxi pri tvorbe aplikácií. Tieto príkazy sú vhodné pre použitie pri tvorbe aplikácií využívajúcej veľké množstvo častí pamäti tzv. buffer-ov pre vysielanie, alebo príjem údajov. Organizáciu pamäti definovanú v „pool-e“ si možno predstaviť ako jednoduchú štruktúru typu jednorozmerného poľa¹¹⁶, kde prvok poľa predstavuje ďalšiu štruktúru s definovaným počtom údajového typu, kde na začiatku je header t.j. popisovač o veľkosti 2 byte informujúci o stave definovaného pool-block. Uvedené funkcie je možné použiť len pri procesoroch s pripojenou externou pamäťou dát XRAM. Vo vyššie uvedenom príklade je sú definované 3 jednoduché pamäťové pool oblasti a jedna zložitejšia štruktúra s názvom „Frame“. Keďže sa jedná o zložitejšiu štruktúru pre skopírovanie jej obsahu do definovaného pool-u je nutné použiť funkciu memcopy().

Obrázok 138, Screenshoot obrazovky pool-managementu RTX51 Full



¹¹⁶ Definované ako pole štruktúr.

Obrázok 139, Správa pamäti v RTX Full cez pool-management

c:\Omega\data\Dropbox\Záloha\C51\I2C by MAXIM in HEX\I2C_by_DSW.c

```
205 struct Ramec {long int Destination,Source; unsigned char Data[10]; int CRC};
206 struct Ramec Frame={ 0x1111,0x2222,1,2,3,4,5,6,7,8,9,10,0x1234};
207 #define Number_Blocks 8
208 #define Block_SizeA 8 //Velkost suvisleho bloku dat 08 byte
209 #define Block_SizeB 16 //Velkost suvisleho bloku dat 16 byte
210 #define Block_SizeC 32 //Velkost suvisleho bloku dat 32 byte
211 #define Block_SizeD sizeof(Frame) //Velkost datovej struktury Frame
212 #define Pool_SizeA (Number_Blocks*(2+Block_SizeA)) //Definujem Number_Blocks blokov
213 #define Pool_SizeB (Number_Blocks*(2+Block_SizeB)) //Definujem Number_Blocks blokov
214 #define Pool_SizeC (Number_Blocks*(2+Block_SizeC)) //Definujem Number_Blocks blokov
215 #define Pool_SizeD (Number_Blocks*(2+Block_SizeD)) //Definujem Number_Blocks blokov
216
217 xdata unsigned char Memory_PoolA[Pool_SizeA];
218 xdata unsigned char Memory_PoolB[Pool_SizeB];
219 xdata unsigned char Memory_PoolC[Pool_SizeC];
220 xdata unsigned char Memory_PoolD[Pool_SizeD];
221 xdata int *Pointers[Number_Blocks]; //Pointre jednotlivych
    blokov ...
222
223 void Init(void) _task_ INIT _priority_ 0
224 {
225     signed char RtxCompletion;
226     unsigned char i;
227     int *Ptr;
228     Watchdog;
229     i=0x00;
230     RtxCompletion=os_create_pool(Block_SizeA,Memory_PoolA,Pool_SizeA); //Definujem "pool" oblast
    kde mam data ...
231     Ptr=os_get_block(Block_SizeA); //Ziaskam pointer na
    premennu v alokovanom pool-e
232     *Ptr=0x1234; //Naplnim premennu v
    alokovanej pamati ....
233     Pointers[i++]=Ptr; //Ulozim do pola pointrov
    ...
234     while ((Ptr=os_get_block(Block_SizeA))!=0) //Ziskavam pointre
    jednotlivych blokov ...
235     {
236         Pointers[i++]=Ptr;
237         *Ptr=0xEDDE;
238     }
239     RtxCompletion=os_create_pool(Block_SizeB,Memory_PoolB,Pool_SizeB); //Definujem "pool"
    oblast kde mam data ...
240     Ptr=os_get_block(Block_SizeB); //Ziaskam pointer na
    premennu v alokovanom pool-e
241     *Ptr=0x5678; //Naplnim premennu v
    alokovanej pamati ....
242     RtxCompletion=os_create_pool(Block_SizeC,Memory_PoolC,Pool_SizeC); //Definujem "pool" oblast
    kde mam data ...
243     Ptr=os_get_block(Block_SizeC); //Ziaskam pointer na
    premennu v alokovanom pool-e
244     *Ptr=0xABCD; //Naplnim premennu v
    alokovanej pamati ....
245     os_free_block(Block_SizeC,Ptr); //Uvolnim pamat premennej
    Ptr v pool-e ...
246
247     Frame.Source=0x89ABCDEF;
248     Frame.Destination=0x01234567;
249     Frame.Data[0]=0xED;
250     Frame.CRC=0xABCD;
251     RtxCompletion=os_create_pool(Block_SizeD,Memory_PoolD,Pool_SizeD);
252     Ptr=os_get_block(Block_SizeD);
253     memcpy(Ptr,&Frame,Block_SizeD);
254     i=0x00;
255     while ((os_free_block(Block_SizeA,Pointers[i++]))==0); //Uvolnujem jednotlivé
    bloky ...
256     Init_Serial();
257     RtxCompletion=os_set_slice(xtal(XTAL));
258     RtxCompletion=os_create_task(THERMO);
259     RtxCompletion=os_create_task(ADC);
260     RtxCompletion=os_delete_task(INIT);
261 }
```


15.27. Management pamäti – odovzdávanie premenných a štruktúr mailbox-om

C:\Omega\data\Dropbox\Záloha\C51\Odovzdávanie štruktúry mailboxom s displayom OK\Odovzdávanie štr

```
1  #include <INC\reg552.h>
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <string.h>
5  #include <rtx51.h>
6
7  extern void Init_Serial(void);
8  extern int ADC_Conversion(unsigned char Channel);
9
10 extern unsigned char Riadok1, Riadok2;
11 extern void LCD_Init(void);
12 extern void LCD_Write_String(char*);
13 extern void LCD_Goto(unsigned char);
14
15 #ifdef SPEED
16     #include <AD\tabulka.inc>
17 #endif
18
19 //-----
20 #define WaitForEver 0xFF
21 //-----
22 #define SYSTEM 0
23 #define CLOCK 1
24 #define DISPLAY 2
25 #define ADC 3
26 //-----
27 #define MBXS 0x00 //Cislomailbox-u
28 #define MBXC 0x01 //Cislomailbox-u
29 //-----
30 extern xdata unsigned char Buffer[BufferSize+1];
31 //-----
32 enum Den {Mon=1, Tue=2, Wed=3, Thu=4, Fri=5, Sat=6, Sun=7};
33 enum Mes {Jan=1, Feb=2, Mar=3, Apr=4, May=5, Jun=6, Jul=7, Aug=8, Sep=9, Oct=10, Nov=11, Dec=12};
34 //-----
35 typedef xdata struct //XDATA je velmi dolezite, lebo IDATA,
DATA nefunguje OK ...
{
36     unsigned char Hod, Min, Sek, Den, Mes, Rok;
37     int Voltage;
38     float Teplota, V;
39 }Time;
40 //-----
41 //-----
42 #define NumMemBlocks 100 //Definujem n pamatovych blokov ...
43 #define SizeMemBlock sizeof(Time)
44 #define SizeMemPool NumMemBlocks*(2+SizeMemBlock)
45 //-----
46 xdata unsigned char Pool[SizeMemPool]; //POOL je niekedy lepsie definovat na
pevnej adrese ale uVision robi problemy pri simulacii ...
47 xdata Time Array = {6, 15, 0, 28, 4, 2013-1900, 0x3FF, +23.0, 1.23};
48 xdata Time *ReceivedMessage; //Adresa struktury prijatej spravy ...
49 xdata unsigned int *MessageToSend; //Adresa struktury vysielanej spravy ...
50 //-----
51
52 void Clock(void) _task_ CLOCK _priority_ 1
53 {
54     signed char RtxCompletion;
55     unsigned int *Ptr; //Ukazovatel v pamati na
56     //odosielanu strukturu ...
57     RtxCompletion=os_create_pool(SizeMemBlock, Pool, SizeMemPool); //Vytvorim pool v pamati XRAM
58     //pre datove struktury ...
59     while(1)
60     {
61         switch(os_wait(K_IVL+K_SIG+K_MBX+MBXC, 100, NULL)) //Cakam na rozne udalosti ...
62         {
63             case SIG_EVENT :
64             case TMO_EVENT : if(++Array.Sek==60)
65             {
66                 Array.Sek=0;
67                 if(++Array.Min==60)
68                 {
69                     Array.Min=0;
70                     if(++Array.Hod==24) Array.Hod=0;
71                 }
72             }
73         }
74     }
75 }
```

C:\Omega\data\Dropbox\Záloha\C51\Odovzdávanie štruktúry mailboxom s displayom OK\Odovzdávanie štr

```
71         os_send_signal (ADC);
72         if ((MessageToSend=os_get_block (SizeMemBlock))==NULL)
73         {
74             os_delete_task (CLOCK); /*Errorhandling*/
75                                     //Rezervujem dynamicku memory ....
76         };
77         Ptr=memcpy (MessageToSend,&Array,sizeof (Array));
78                                     //Skopirujem strukturu Array do struktury MessageToSend
79         a necham inicializovany ukazovatel ...
80         RtxCompletion=os_send_message (MBXS,(unsigned int)MessageToSend,WaitForEver
81 );                                     //Posielam len adresu dat ulozenych v strukture mailboxom ...
82         if ((RtxCompletion=os_free_block (SizeMemBlock,MessageToSend))!=NULL)
83         {
84             os_delete_task (CLOCK); /*Errorhandling*/
85                                     //Uvolnim dynamicku memory ....
86         }
87         break;
88     case MSG_EVENT : break;
89 }
90 }
91
92 void Display (void) _task_ DISPLAY _priority_ 0
93 {
94     signed char RtxCompletion;
95     LCD_Init ();
96     while (1)
97     {
98         switch (RtxCompletion=os_wait (K_SIG+K_TMO+K_INT+K_MBX+MBXS,WaitForEver,(unsigned int xdata*)&
99 ReceivedMessage))
100     {
101         case TMO_EVENT :
102         case INT_EVENT : //RtxCompletion=0;
103         case SIG_EVENT :
104         case MSG_EVENT : sprintf (Buffer,"%02bd:%02bd:%02bd,V=%2.2fv\r\n",ReceivedMessage->Hod,
105 ReceivedMessage->Min,ReceivedMessage->Sek,ReceivedMessage->V);
106         printf ("%s",Buffer);
107         LCD_Goto (Riadok2);
108         LCD_Write_String (Buffer);
109         LCD_Goto (Riadok1);
110         strcpy (Buffer,"DSWSoftware\r\n");
111         LCD_Write_String (Buffer);
112         break;
113     }
114 }
115
116 void Adc (void) _task_ ADC _priority_ 0
117 {
118     signed char RtxCompletion;
119     TCON |=0x01+0x04; //Spustim /INT0 a /INT1 hranou a nie urovnou !!!
120     os_attach_interrupt (0); //Pripojim sa na prerusenie /INT0
121     os_attach_interrupt (2); //Pripojim sa na prerusenie /INT1
122     while (1)
123     {
124         switch (RtxCompletion=os_wait (K_TMO+K_SIG+K_INT,WaitForEver,NULL))
125         {
126             case INT_EVENT :
127             case SIG_EVENT :
128             case TMO_EVENT : Array.Voltage=ADC_Conversion (0);
129                             #ifdef SPEED
130                             Array.V=Tab [Array.Voltage];
131                             #else
132                             Array.V=(5.0/1024.0)*Array.Voltage; break; //Prevediem AD hodnotu na int
133                             a float ...
134                             #endif
135         }
136     }
137 }
138
139 void System (void) _task_ SYSTEM _priority_ 0
140 {
141     signed char RtxCompletion;
142     RtxCompletion=os_set_slice (15360); // 15360=10ms os_wait, 1536=1ms os_wait
```

C:\Omega\data\Dropbox\Záloha\C51\Odovzdávanie štruktúry mailboxom s displayom OK\Odovzdávanie štr

```
137  RtxCompletion=os_create_task (ADC);
138  RtxCompletion=os_create_task (CLOCK);
139  RtxCompletion=os_create_task (DISPLAY);
140  Init_Serial ();
141  while (1)
142  {
143      RtxCompletion=os_wait (K_TMO, 50, NULL);
144      os_delete_task (SYSTEM);
145  }
146  }
147
148  void main (void)
149  {
150      signed char RtxCompletion;
151      RtxCompletion=os_start_system (SYSTEM);
152  }
```

V praxi je niekedy potrebné inej úlohe odovzdať¹¹⁷ nielen jednoduchú premennú, ale aj zložitejšiu štruktúru. Vyššie uvedený program ukazuje možnosť ako je možno odovzdať štruktúru a jej obsah ďalšej úlohe. Pomocou zápisu v jazyku C môžeme prijať pomocou mailbox-u aj pomerne zložitú štruktúru¹¹⁸. Štruktúru je možné chápať v podstate ako premennú v Pascal-e známu ako typ pole t.j. „array“ ktorá je alokovaná v priestore pamäti externej pamäti označovanej ako XDATA.

Alokovanie pamäti pre štruktúru je možné pomocou príkazu `os_create_pool()` t.j. `RtxCompletion=os_create_pool(SizeMemBlock,Pool,SizeMemPool)`, kde **SizeMemBlock** predstavuje veľkosť potrebnej pamäti štruktúry, ukazovateľ **Pool** už predstavuje samotnú štruktúru uloženú v externej pamäti ktorá môže byť už aj ako **n** počet blokov a **SizeMemPool** udáva celkovú výslednú veľkosť samotnej štruktúry.

`switch(RtxCompletion=os_wait(K_SIG+K_TMO+K_INT+K_MBX+MBXS,WaitForEver, (unsigned int xdata*)&ReceivedMessage));` Výsledky odovzdané štruktúrou môžeme potom použiť v príkaze `printf()`, alebo `sprintf()` čo je vyjadrené v nasledujúcom zápise:

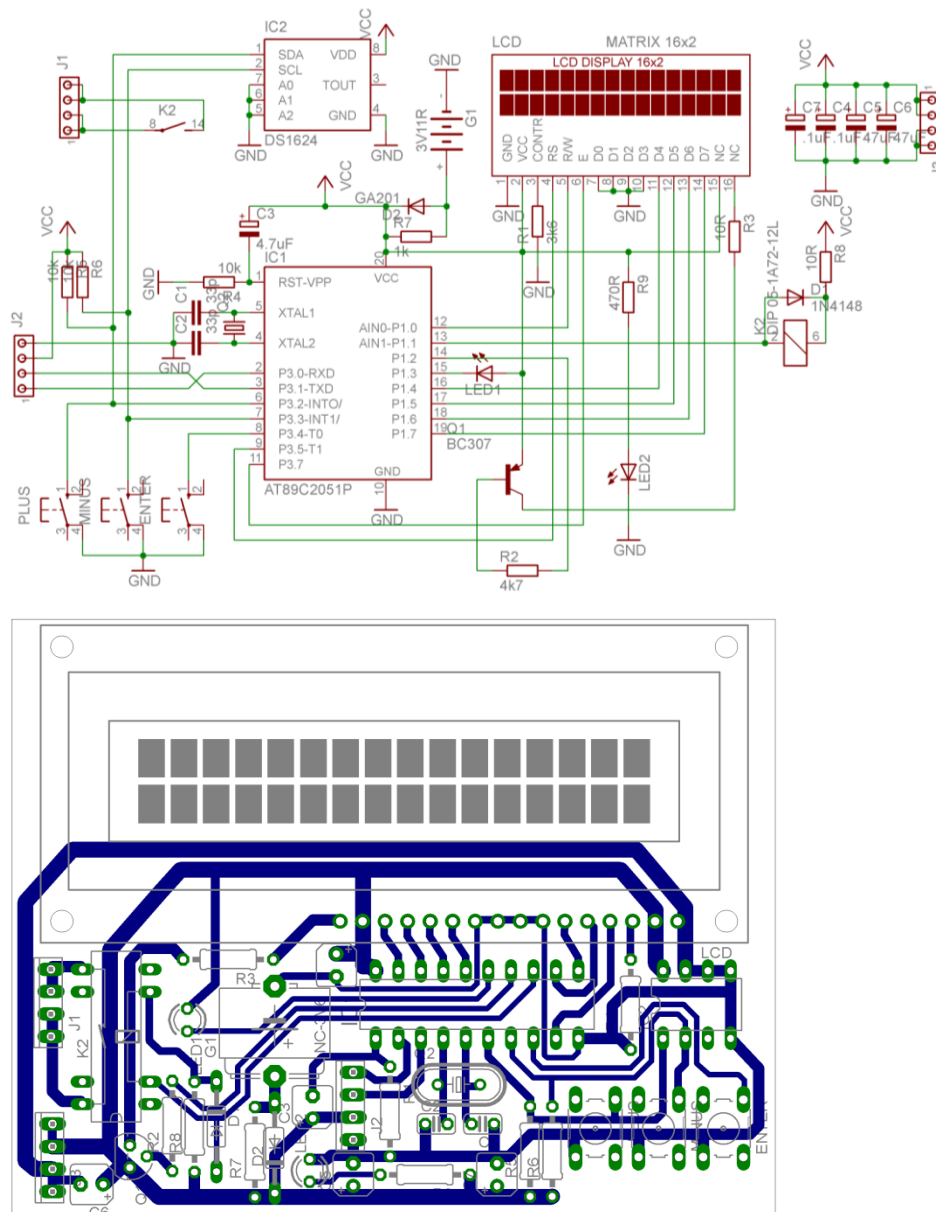
`sprintf(Buffer,"%02bd:%02bd:%02bd,V=%2.2fV\r\n",ReceivedMessage->Hod,ReceivedMessage->Min,ReceivedMessage->Sek,ReceivedMessage->V);`

¹¹⁷ RTX51 umožňuje odovzdať pomocou premennej mailbox-u maximálne premennú **int**, čo v našom prípade môže byť nielen premenná, ale aj ukazovateľ ktorý umožní odovzdať aj adresy zložitejších štruktúr a objektov.

¹¹⁸ V μ Vision4 9.50 je v okne WATCH1 náhodne nesprávne zobrazovaný obsah samotnej štruktúry. V tomto prípade obsahuje štruktúra pri simulácii menej prvkov štruktúry ako je definované a ich obsah je nesprávny. V μ Vision3 je štruktúra vždy správne zobrazovaná so správnym počtom prvkov štruktúry. Chyba je spôsobená pravdepodobne zlou počiatočnou inicializáciou samotných vlastností objektov programu μ Vision4. Vo verzii 9.52.0.0 μ Vision4 sa chyba už nevyskytuje.

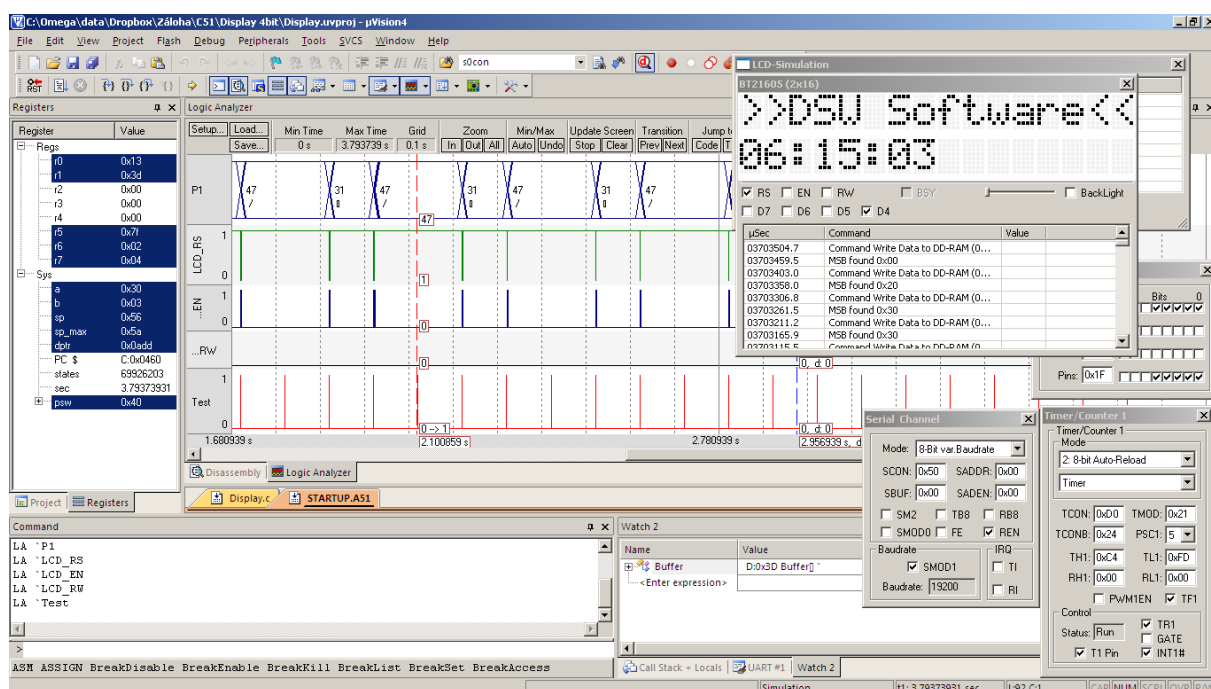
15.28. Ovládanie radiča HD44780 LCD display-a s RTX51 Tiny.

Radiče HD44780 od svojho začiatku sa vzhľadom na svoju jednoduchosť ovládania ihneď stali veľmi obľúbenými prvkami všetkých elektronických zariadení. Programátor môže ovládať¹¹⁹ radič display-a LCD jednotky z pomocou 4, alebo 8 bitovej zbernice a 3 základných riadiacich signálov RS, RW a E.



¹¹⁹ Ovládanie display-a cez AGSI ovládače v prostredí μ Vision5 je možné na všetkých portoch mikroprocesora pomocou 4 bitovej zbernice, pričom musia byť jednotlivé komunikačné vodiče deklarované ako premenná typu bit a nie byte! Napr. správna definícia `sbit LCD_0=P1^0`; Najčastejšie sa robia chyby ak sa priamo posielajú dáta pomocou príkazov priameho výstupu na port `P1=(0x33&0x0F)`; Vtedy sa nezobrazujú údaje zapísané priamo do portu P1 na display cez AGSI interface. V súbore `display.Lcd` musí byť interným premenným AGSI display-a `Dx_NAME` priradená priamo premenná napr. `D4_NAME=LCD_4` t.j. `sbit LCD_4=P4^4` ktorá je použitá v programe a nie priamo port P4, lebo v tomto prípade LCD display nedokáže správne zobrazovať údaje.

Obrázok 140, Simulácia LCD display-a v prostredí Keil μ Vision4



Na simuláciu LCD sme použili AGSI¹²⁰ ovládače nachádzajúce sa voľne dostupné na internete. Jediným obmedzením je možnosť simulácie len jedného LCD radiča pomocou 4 bitovej komunikácie. Navyše všetky dátové vodiče musia byť v programe deklarované ako bitové premenné s veľkosťou jedného bitu tzv. sbit. V prípade použitia rýchlejšej varianty AT89LP4052 namiesto AT89C4051 je nutné ešte porty nastaviť ako obojsmerné¹²¹. Po resete sú všetky nastavené ako vstupné. Zároveň sa musí v RTX51 Tiny nastaviť konštanta INT_CLOCK na 12-násobne vyššiu hodnotu, pretože AT89LP4052 je procesor tzv. single-cycle, rovnako aj parametre sériového komunikačného rozhrania RS232.

Vyššie uvedená aplikácia prijíma údaje zo sériového rozhrania RS232¹²² a prijatý reťazec procesor zobrazí na display-i. Príjem údajov je zabezpečený výskytom tzv. komunikačného tokenu "^\n" za ktorým nasleduje číslo zobrazovaného riadku (1,2...n) display-a a nakoniec samotný zobrazovaný reťazec ktorý je ukončený znakom "\n". V prípade, že bude ukončovaci znak chýbať, zobrazí sa maximálne len 16 znakov na jeden riadok LCD. Ostatné prijaté znaky sú ignorované.

¹²⁰ <http://www.c51.de/c51.de/Dateien/uVision2DLLs.php?Spr=EN>

¹²¹ P1M0=P3M0=P1M1=P3M1=0x00;

¹²² Signály RXD a TXD na porte P3.0 a P3.1 procesora AT89LP4052.

C:\Omega\data\Dropbox\Záloha\C51\Display4bit\Display.c

```
1  #include <stdio.h>          /* standard I/O .h-file          */
2  #include <ctype.h>          /* character functions      */
3  #include <string.h>         /* string and memory functions */
4  #include <stdlib.h>
5  #include <at89LP4052.h>
6  #include <rtx51tny.h>
7  #include <dsw.h>
8
9  #include <adresy.inc>
10
11 #define SYSTEM 0
12 #define CLOCK 1
13 #define SERIAL 2
14 #define ROTATE 3
15
16 #define Port P1
17 #define LCD 0xFF //Inicializacia portu ....
18 #ifdef x12
19     #define Dlzka 255 //pre 18.432MHz AT89LP4052
20 #else
21     #define Dlzka 60 //pre 18.432MHz
22 #endif
23 #define Riadok1 0x00
24 #define Riadok2 0x40
25
26 sbit LCD_RW = P3^4;
27 sbit LCD_EN = P3^7;
28 sbit LCD_RS = P3^5;
29
30 extern void Delay (unsigned char);
31 extern void Init_Serial (void);
32
33 sbit LCD7 = P1^7;
34 sbit LCD6 = P1^6;
35 sbit LCD5 = P1^5;
36 sbit LCD4 = P1^4;
37
38 unsigned char Buffer [16+1];
39
40 void LCD_Clock (void)
41 {
42     LCD_EN=1;
43     #ifdef x12 //Cakaj cca. 50us
44         Delay (Dlzka);
45         Delay (Dlzka);
46         Delay (Dlzka);
47     #else //Cakaj cca. 50us
48         Delay (Dlzka);
49     #endif
50     LCD_EN=0;
51 }
52
53 void LCD_Send (unsigned char Data)
54 {
55     LCD7=(Data &0x80)?1:0; //Posielam MSB
56     LCD6=(Data &0x40)?1:0;
57     LCD5=(Data &0x20)?1:0;
58     LCD4=(Data &0x10)?1:0;
59     LCD_Clock ();
60     LCD7=(Data &0x08)?1:0; //Posielam LSB
61     LCD6=(Data &0x04)?1:0;
62     LCD5=(Data &0x02)?1:0;
63     LCD4=(Data &0x01)?1:0;
64     LCD_Clock ();
65 }
66
67 void LCD_Data (unsigned char Data)
68 {
69     LCD_RS=1; //Pretoze odosielam data !!!
70     LCD_EN=0;
71     LCD_RW=0;
72     LCD_Send (Data);
73 }
74
```

Page1

C:\Omega\data\Dropbox\Záloha\C51\Display4bit\Display.c

```
75 void LCD_Cmd (unsigned char Data)
76 {
77     LCD_RS = 0;           // Pretože odosielať príkaz !!!
78     LCD_EN = 0;
79     LCD_RW = 0;
80     LCD_Send (Data);
81 }
82
83 void LCD_ClrScr (void)
84 {
85     LCD_Cmd (0x01);
86     os_wait (K_TMO, 2, NULL);
87 }
88
89 void LCD_Goto (unsigned char Ptr)
90 {
91     LCD_Cmd (0x80 + Ptr);
92     Delay (Dlžka);
93 }
94
95 void LCD_Home (void)
96 {
97     LCD_Cmd (0x02);
98     os_wait (K_TMO, 2, NULL);
99 }
100
101 void LCD_Init (void)
102 {
103     LCD_RS = 0;
104     LCD_RW = 0;
105     LCD_EN = 0;
106     #ifdef x12
107         P1M0 = P3M0 = P1M1 = P3M1 = 0x00;
108     #endif
109     Port = LCD;
110     os_wait (K_TMO, 20, NULL); // Čakaj cca. 20ms
111     LCD_Cmd (0x33);
112     os_wait (K_TMO, 5, NULL);
113     LCD_Cmd (0x32);
114     os_wait (K_TMO, 5, NULL);
115     LCD_Cmd (0x20 + 0x08);
116     LCD_Cmd (0x10 + 0x00);
117     LCD_Cmd (0x08 + 0x04);
118     LCD_Cmd (0x04 + 0x02);
119     LCD_ClrScr ();
120     LCD_Home ();
121 }
122
123 void LCD_Write_String (char *String)
124 {
125     register unsigned char i;
126     for (i = 0x00; i <= sizeof (Buffer); i++)
127     {
128         switch (*String)
129         {
130             case '\n' :
131             case '\r' :
132             case '\0' : *String = ' '; break;
133         }
134         LCD_Data (*String++);
135     }
136     memset (Buffer, ' ', sizeof (Buffer));
137 }
138
139 #ifdef SIMUL
140
141 struct Hodiny
142 {
143     unsigned char Hod, Min, Sek, Des;
144 };
145
146 struct Hodiny Cas = {6, 15, 0, 0};
147
148 void Clock (void) _task_ CLOCK
```

Page2

C:\Omega\data\Dropbox\Záloha\C51\Display4bit\Display.c

```
149 {
150     while (1)
151     {
152         os_wait (K_IVL, 100, NULL);
153         os_send_signal (SYSTEM);
154         if (++Cas.Des == 10)
155         {
156             Cas.Des = 0;
157             if (++Cas.Sek == 60)
158             {
159                 Cas.Sek = 0;
160                 if (++Cas.Min == 60)
161                 {
162                     Cas.Min = 0;
163                     if (++Cas.Hod == 24)
164                     {
165                         Cas.Hod = 0;
166                     }
167                 }
168             }
169         }
170     }
171 }
172
173 #endif
174
175 #ifdef INT232
176
177 void IntrS232 (void) interrupt 4 //using 3
178 {
179     switch (SCON & 0x03)
180     {
181         case 0x01 : RI=0; isr_send_signal (SERIAL); break;
182         case 0x02 : TI=0; break;
183         case 0x03 : TI=RI=0; break;
184     }
185 }
186
187 unsigned char Addr;
188
189 void Serial (void) _task_ SERIAL //Otestovane a ide OK
190 {
191     bit RECEIVE;
192     register unsigned char i;
193     ES=1;
194     while (1)
195     {
196         os_wait (K_SIG, 0xFF, NULL);
197         if (Token == SBUF)
198         {
199             os_wait (K_SIG, 0xFF, NULL);
200             switch (Addr = SBUF)
201             {
202                 case Adresa1 :
203                 case Adresa2 :
204                 case Adresa3 :
205                 case Adresa4 : RECEIVE=0; i=0x00; //Prijal som platnu adresu ...
206                     while (!RECEIVE)
207                     {
208                         os_wait (K_SIG, 0xFF, NULL);
209                         switch (Buffer[i++] = SBUF) //Citam SERIAL az pokial nenajdem koniec
210                         {
211                             case '\r' :
212                             case '\n' :
213                             case '\0' : RECEIVE=1; os_send_signal (SYSTEM); break;
214                         }
215                     }
216                     break;
217                 default : break;
218             }
219         }
220     }
221 }
```

Page3

C:\Omega\data\Dropbox\Záloha\C51\Display4bit\Display.c

```
222
223 #else
224
225 unsigned char Readkey (void) //Otestovane a ide OK
226 {
227     while (!RI) os_switch_task ();
228     RI=0;
229     return (SBUF);
230 }
231
232 unsigned char Addr;
233
234 void Serial (void) _task_ SERIAL
235 {
236     bit RECEIVE;
237     register unsigned char i;
238     ES=0;
239     while (1)
240     {
241         if (Token == Readkey ())
242         {
243             switch (Addr=Readkey ())
244             {
245                 case Adresa1 :
246                 case Adresa2 :
247                 case Adresa3 :
248                 case Adresa4 : RECEIVE=0; i=0x00; //Prijal som platnu adresu ...
249                     while (!RECEIVE)
250                     {
251                         switch (Buffer[i++]=Readkey ()) //Citam SERIAL az pokial nenajdem koniec
252                         {
253                             case '\r' :
254                             case '\n' :
255                             case '\0' : RECEIVE=1; os_send_signal (SYSTEM); break;
256                         }
257                     }
258                     break;
259                 default : break;
260             }
261         }
262     }
263 }
264
265 #endif
266
267 #ifdef ROTATESTRING
268
269 xdata unsigned char Buf [100];
270
271 void Rotate (void) _task_ ROTATE
272 {
273     register unsigned char Znak, Length;
274     while (1)
275     {
276         os_wait (K_TMO, 250, NULL);
277         Length=strlen (Buf)-1;
278         Znak=Buf [0];
279         memmove (Buf, Buf+1, Length);
280         Buf [Length]=Znak;
281     }
282 }
283
284 #endif
285
286 code unsigned char Test []="!!!>>>TEST<<<!!!";
287
288 void System (void) _task_ SYSTEM
289 {
290     code unsigned char Logo []=">>DSW Software<<";
291     #ifdef SIMUL
292     #ifdef DATE
293         code unsigned char *SystemTime[]={__TIME__, __DATE__, __DATE2__, __C51__};
294         sscanf (SystemTime [0], "%02bd:%02bd:%02bd", &Cas .Hod, &Cas .Min, &Cas .Sek);
295     }
296 }
```

Page4

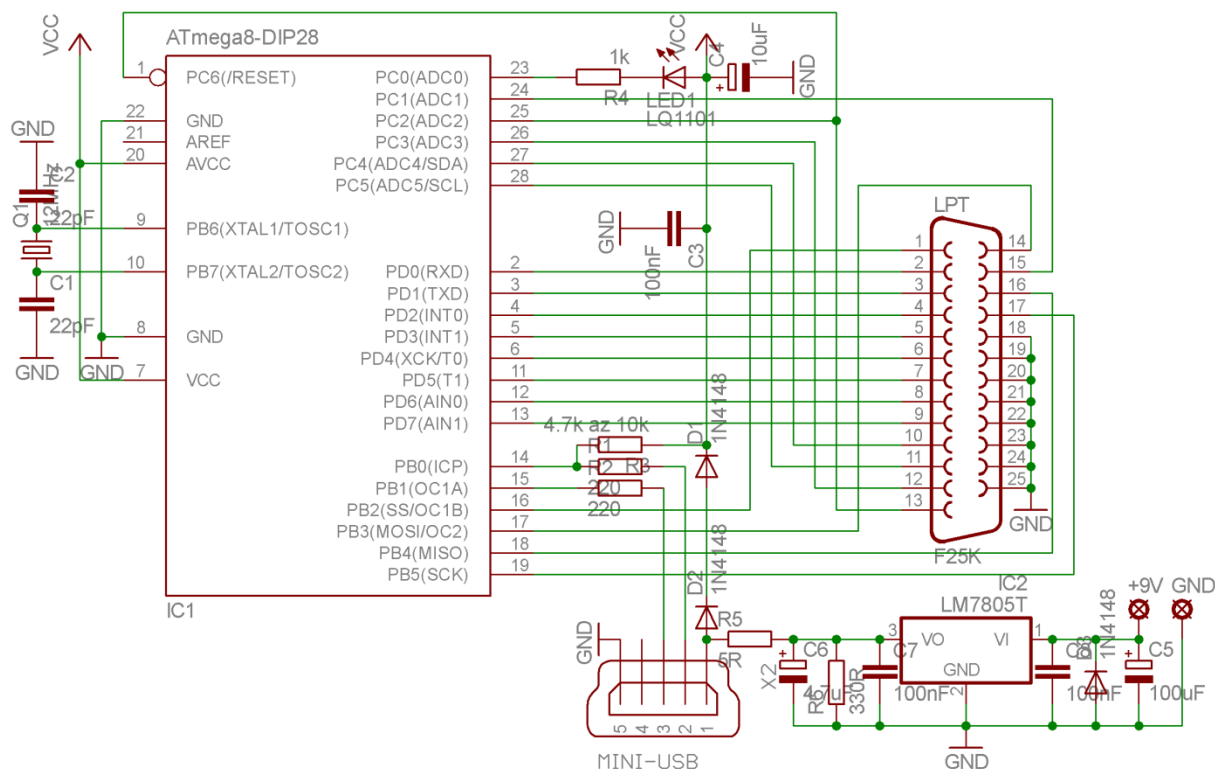
C:\Omega\data\Dropbox\Záloha\C51\Display4bit\Display.c

```
295     #endif
296     #endif
297     Init_Serial ();
298     LCD_Init ();
299     #ifdef SIMUL
300         os_create_task (CLOCK);
301     #endif
302     os_create_task (SERIAL);
303     #ifdef ROTATESTRING
304         os_create_task (ROTATE);
305     sprintf (Buf, "%s", "Diversant Software (c) 2013, Ing. Eduard Jadron, Stefana Nahalku 18/65, 036
01 Martin, Slovakia, 0123456789ABCDEF, ");
306     #endif
307     #ifndef SIMUL
308         LCD_Goto (Riadok1);
309         sprintf (Buffer, "%s", "Copyright (c) 2013");
310         LCD_Write_String (Buffer);
311         LCD_Goto (Riadok2);
312         sprintf (Buffer, "%s", Logo);
313         LCD_Write_String (Buffer);
314     #endif
315     while (1)
316     {
317         switch (os_wait (K_SIG+K_TMO, 0xFF, NULL))
318         {
319             case TMO_EVENT : sprintf (Buffer, "%s", "Serial Error !!!");
320                             LCD_Goto (Riadok1);
321                             LCD_Write_String (Buffer);
322                             break;
323             case SIG_EVENT :
324                             #ifdef SIMUL
325                             if (Addr++> 2) Addr=1;
326                             #endif
327                             switch (Addr)
328                             {
329                                 case Adresa1 : LCD_Goto (Riadok1);
330                                                 #ifdef SIMUL
331                                                 sprintf (Buffer, "%s", Logo);
332                                                 #ifdef ROTATESTRING
333                                                 memcpy (Buffer, Buf, sizeof (Buffer)-1);
334                                                 #endif
335                                                 #endif
336                                                 LCD_Write_String (Buffer);
337                                                 break;
338                                 case Adresa2 : LCD_Goto (Riadok2);
339                                                 #ifdef SIMUL
340                                                 sprintf (Buffer, "%02bd:%02bd:%02bd", Cas .Hod, Cas .Min, Cas .
341                                                 Sek );
342                                                 #endif
343                                                 LCD_Write_String (Buffer);
344                                                 break;
345                                 case Adresa3 : break;
346                                 case Adresa4 : break;
347                             }
348             }
349     }
```

15.29. Prevedník portu USB na LPT (USB2LPT)

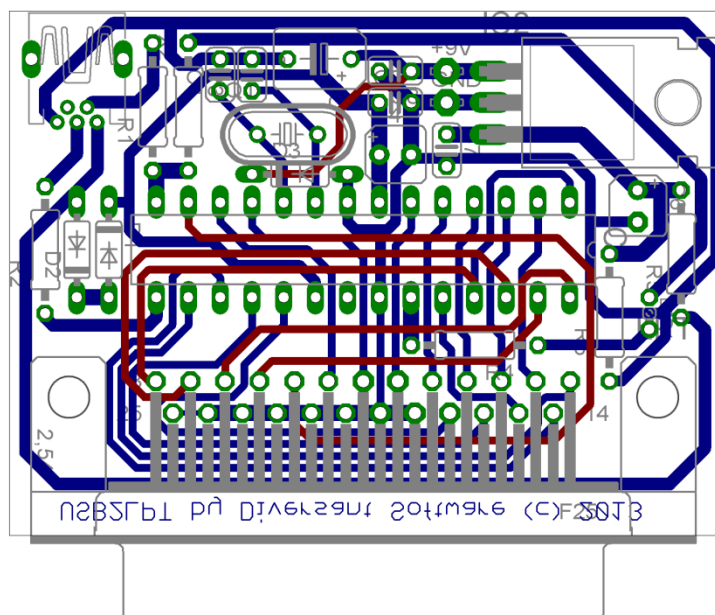
Dnešné moderné počítače alebo notebooky už vo svojej výbave nemajú LPT port známy pod názvom CENTRONICS, alebo IEEE1284. Donedávna slúžil na pripojenie tlačiarň, ale príchodom USB bol vytlačený ako neperspektívne rozhranie. Niektoré aplikácie, ale hlavne hardware-ové zariadenia ho dodnes používajú. Príkladom sú staršie programátory od firmy ELNEC. Na trhu nie sú špecializované prevodníky z USB na LPT a tak je potrebné vytvoriť rozhranie USB2LPT aby bolo možné používať tieto zariadenia aj naďalej. Na trhu síce existujú hotové prevodníky z USB na LPT špeciálne určené pre tlačiarne, ale tieto sú absolútne nevhodné, pretože umožňujú komunikovať len s ovládačmi tlačiarne a operačného systému. Nemajú priradenú komunikačnú adresu portu LPT (378H, 278H, 3BCH) a príslušné IRQx prerušenie. I keď je USB prevádzkované s prenosovou rýchlosťou 480Mbps zmena z režimu čítanie/zápis trvá min. 125μs oproti 1μs v prípade LPT portu. Z tohto dôvodu nie je jednoduché urobiť jednoduchý prevodník ktorý by cez USB obsluhoval LPT port. Jednoducho sa dá podobné zariadenie vyrobiť s pomocou mikroprocesorom ATmega8 a niekoľkých súčiastok. Software vrátane zdrojových programov a hardware je neustále dostupné na stránkach pod názvom USB2LPT.¹²³

Obrázok 141, Schéma zapojenia USB2LPT portu



¹²³ <http://www-user.tu-chemnitz.de/~heha/bastelecke/Rund%20um%20den%20PC/USB2LPT/index.html.en>

Obrázok 142, Doska plošného spoja USB2LPT portu



15.30. PWM kanál s AT89C4051

Mikroprocesor AT89C4051 neobsahuje PWM kanál ktorý je možné využiť napríklad na ovládanie jasú LED diódy, prípadne na budenie tranzistora ktorý ovláda jednosmerný obvod. Až u novších procesoroch si u firmy ATMEL uvedomili, skutočný význam PWM kanála a od AT89LP4052¹²⁴ je už zabudovaná dvojica PWM kanálov. Obrázok 143 upresňuje princíp fungovania PWM kanála. Základ PWM generátora je vytvorený časovačom T0 ktorý je nastavený do 8 bitového „autoreload“ režimu s periodickým vyvolávaním prerušenia po pretečení T0. Premenné t_{ON} a t_{OFF} obsahujú dĺžku intervalov zopnutia tranzistorov LED display-a. Tento spôsob maskovania je ideálne použiteľný na generovanie PWM signálu pre 7-segmentový display. Premenná „Bright“ v programe predstavuje hodnotu PWM veličiny, ktorou sme schopný regulovať jas LED display-a.

¹²⁴ Pre zaistenie kompatibility mikroprocesora AT89LP4052 s AT89C4051 je nevyhnutné nastaviť I/O konfiguračné registre $P1M0=P3M0=P1M1=P3M1=0x00$. Nastavenie je nevyhnutné kvôli výstupnému režimu portu. V prípade, že používame sériové komunikačné rozhranie RS232 je nutné si uvedomiť, že sa jedná o single-cycle procesor ktorý je cca. 12x rýchlejší a tiež musíme nastaviť iné hodnoty generátorov prenosových rýchlostí v $TH1=TL1=(0x100-(((long int)XTAL/16*1)/BAUD));$

Obrázok 143, Generovanie PWM signálu u AT89C4051 pomocou časovača T0

```
1  #include <reg51.h>
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <dsw.h>
5  #include <adresy.inc>
6
7  extern code char Velkost ;
8  extern code unsigned char Tab [];      //Tabulka s 7 segmentovymi cislami
9
10 enum {CN=1, OFF=0};
11
12 #ifdef BRIGHT
13
14 bit Light ;                          //Definujem bit svietenia pre PWM_LED
15 unsigned char Tmp,Bright,ton,toff;
16
17 void PWM0(void) interrupt 1 using 2    //Generujem PWM len s jedným časovačom T0
18 {
19     ET0=0;
20     TR0=OFF;                          //Vypnem časovac T0
21     if (Light)
22     {
23         TH0=toff;                     //Budem generovať periodu vypnutia
24         P1=0x00;                       //Vypnem zobrazovanie len na niektorých pinoch ...
25         Light=OFF;                     //Nastavím príznak svietenia
26     }
27     else
28     {
29         TH0=ton;                       //Nastavím požadovanú hodnotu jasu
30         P1=Tmp;                         //Obnovím hodnotu na porte P1 len na niektorých pinoch ...
31         Light=CN;                       //Nastavím príznak svietenia
32     }
33     TR0=CN;                            //Zapnem časovac
34     ET0=1;
35 }
36
37 void Led(unsigned char Cislo)
38 {
39     TR0=OFF;
40     P1=Tmp=Tab[Cislo];                //Prevediem číslo na LED segment
41     EA=1;
42     ET0=1;                             //Povolím prerušenie od T0
43     Light=CN;                           //Nastavím príznak svietenia
44     ton=0xFF-Bright;                    //Nastavím požadovanú hodnotu jasu
45     toff=Bright;                         //Budem generovať periodu vypnutia
46     TH0=toff;                           //Nastavím požadovanú hodnotu jasu
47     TR0=CN;                             //Spustím časovac
48 }
49
50 #else
51
52 void Led(unsigned char Cislo)
53 {
54     P1=Tab[Cislo];                      //Prevediem číslo na LED segment
55 }
56
57 #endif
58
```

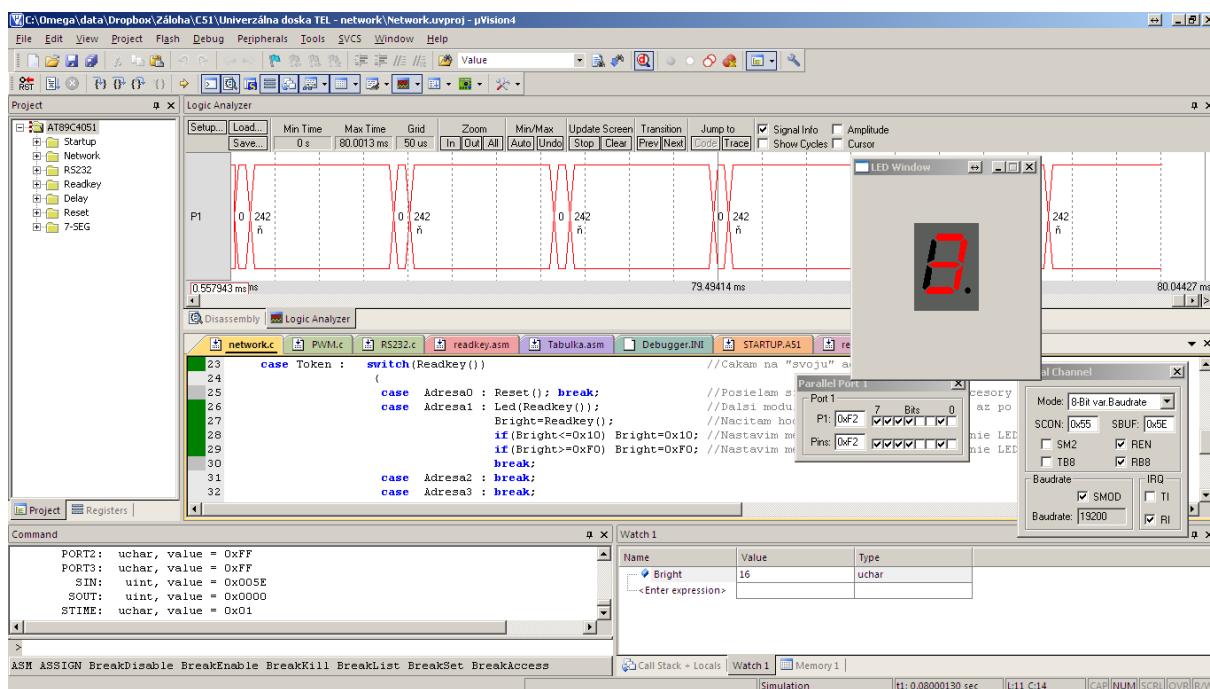
Premenná **Tmp** obsahuje hodnotu vyslanú na port P1. Ak by sme čítali port P1 tak by sme čítali priamo vývody mikroprocesora ktoré sú pripojené priamo na bázu tranzistora čo by nám v konečnom dôsledku prečítalo¹²⁵ nesprávnu hodnotu pretože zvyšovacie rezistory na vývodoch nám ovplyvnia logickú hodnotu na jednotlivých vývodoch P1.

¹²⁵ Pre správnu hodnotu by sme mali čítať záchytný klopný obvod portu P1 mikroprocesora, kde sú zapísané hodnoty do portu a ne logické hodnoty nachádzajúce sa na jednotlivých vývodoch portu ktoré môžu byť skreslené.

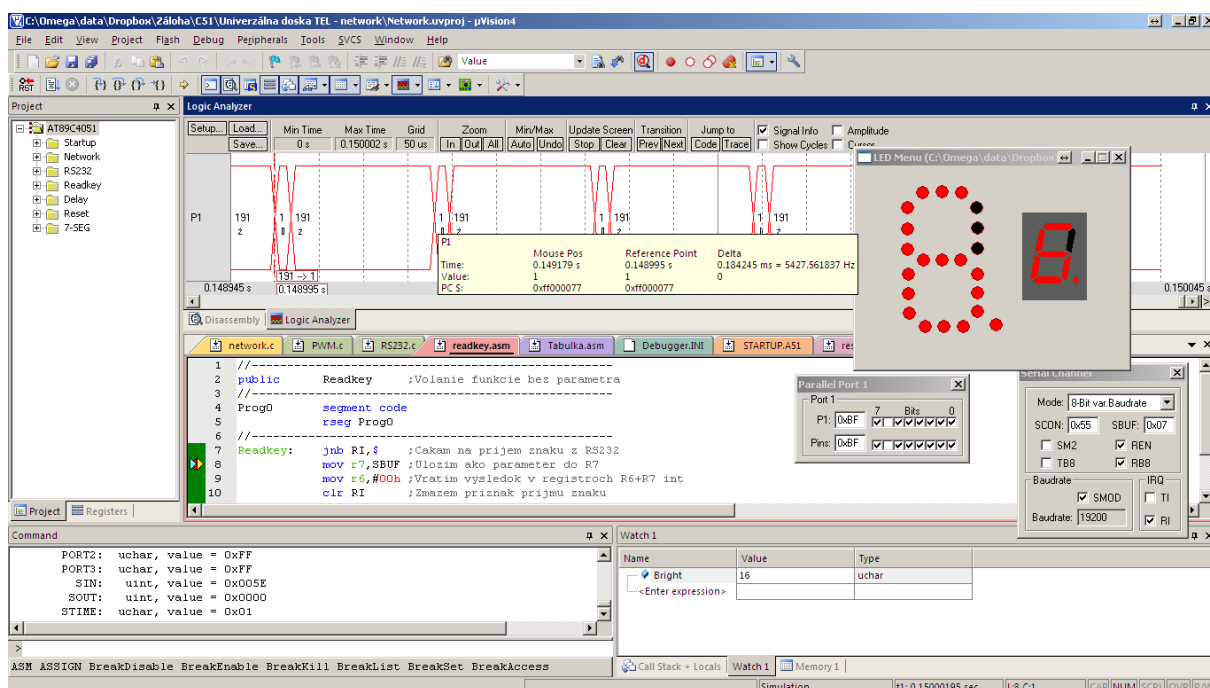
Obrázok 144, Prevodná tabuľka 7 seg-mentového displaya v assembleri A51

```
1  ;-----
2  name      Tabulka
3  ;-----
4  public    Tab, Velkost ;
5  ;-----
6  Prog0     segment code inpage ;segment umiestneny v ramci jednej 256-byte stranky
7           rseg Prog0
8  ;-----
9  ;          .abcdefg      ;Jednotlive segmenty LED display-a
10 Tab :
11         db 01111110b ;0
12         db 00110000b ;1
13         db 01101101b ;2
14         db 01111001b ;3
15         db 00110011b ;4
16         db 01011011b ;5
17         db 01011111b ;6
18         db 01110000b ;7
19         db 01111111b ;8
20         db 01111011b ;9
21         db 01110111b ;A
22         db 00011111b ;B
23         db 00001101b ;C
24         db 00111101b ;D
25         db 01001111b ;E
26         db 01000111b ;F
27         db 01011110b ;0
28         db 00010111b ;H
29         db 00000100b ;I
30         db 00111100b ;J
31         db 00000111b ;K
32         db 00001110b ;L
33         db 01110110b ;M
34         db 00010101b ;N
35         db 00011101b ;O
36         db 01100111b ;P
37         db 00110111b ;X
38         db 0000001b  ;-
39
40 ;-----
41 Velkost   equ ($-Tab)      ;Velkost:   db ($-Tab)
42 ;-----
43 end
```

Obrázok 145, Simulácia PWM kanála so 7-segmentovým displayom a LED display-om



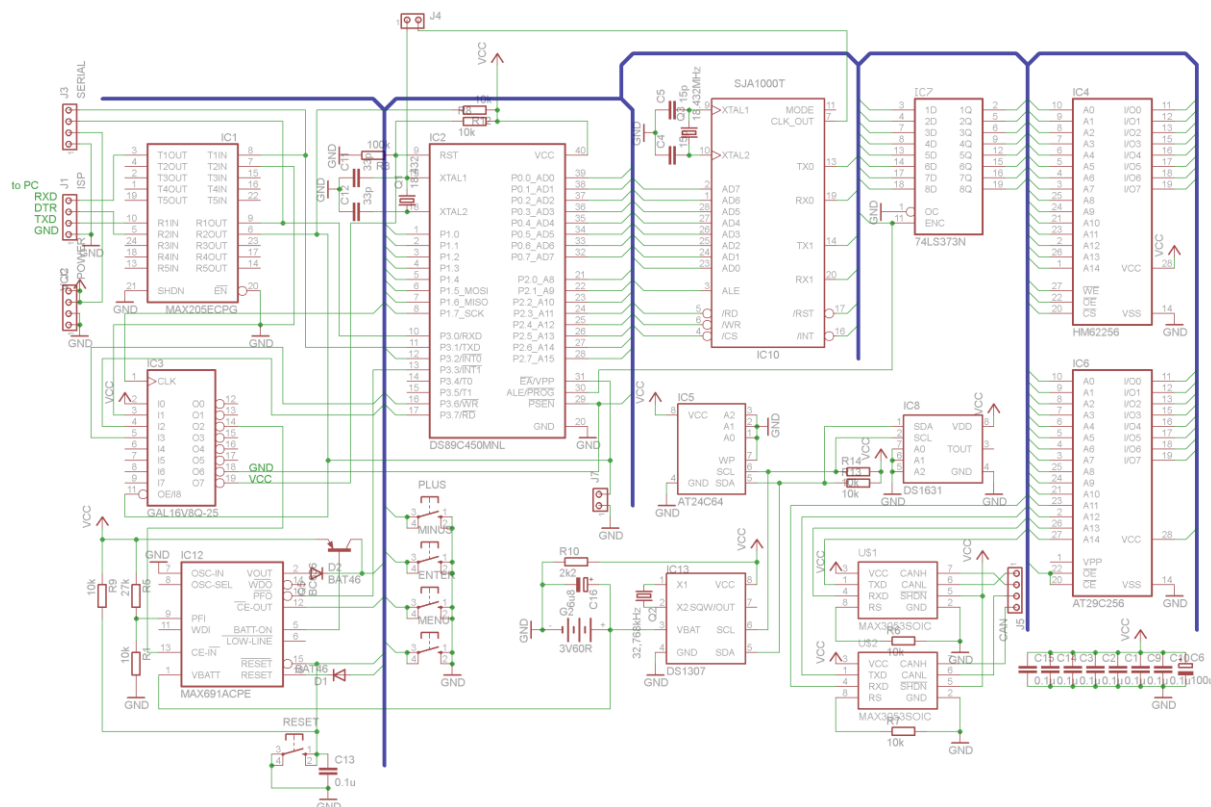
Obrázok 146, Simulácia PWM kanála s LED display-om



15.31. Univerzálne komunikačné rutiny s CAN rozhraním

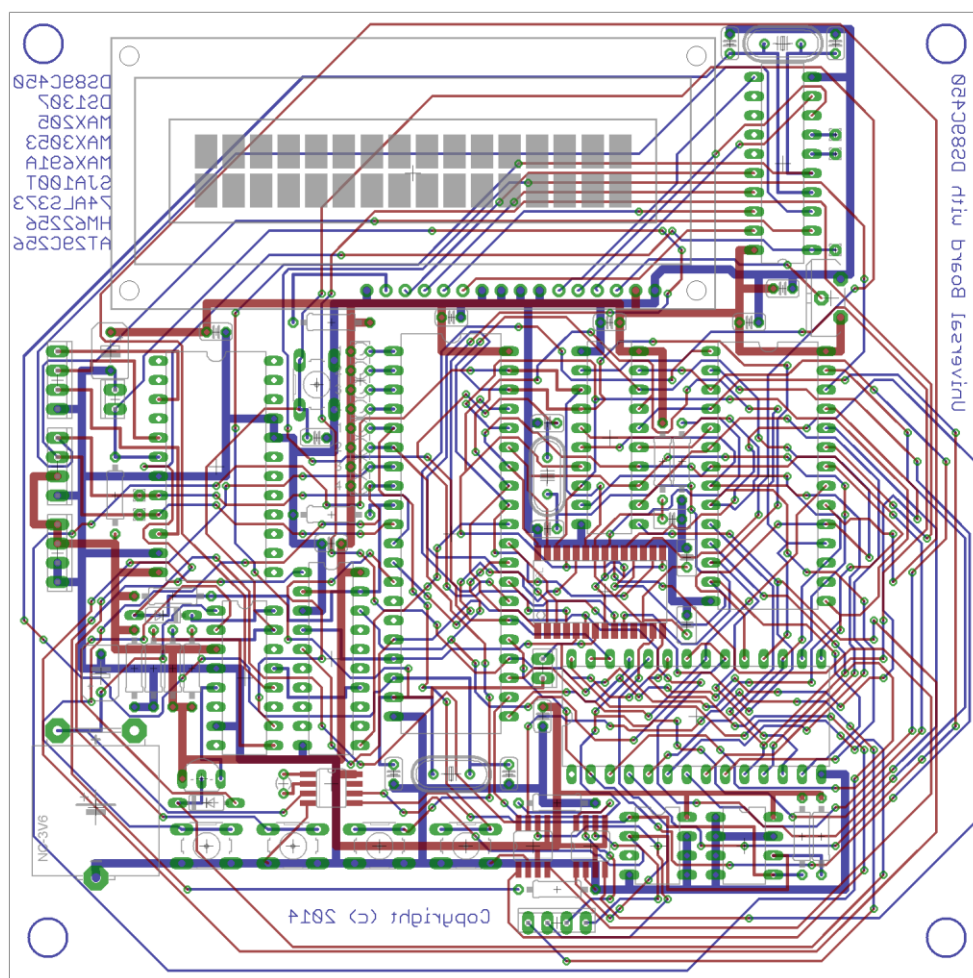
Komunikačné rozhranie CAN definovala v roku 1986 firma Bosch pre potreby automobilového priemyslu. Vďaka jej vynikajúcim vlastnostiam sa dostala aj do spotrebičov každodennej spotreby. Podpora¹²⁶ CAN je samozrejmosťou u RTX51 v PK51 pre špeciálne vyvinuté obvody SJA1000T, Philips 82C200, Philips 80592, Siemens c515C, Siemens 81C90 a Intel 82527.

Obrázok 147, Zapojenie univerzálneho modulu s DS89C450 s podporou CAN

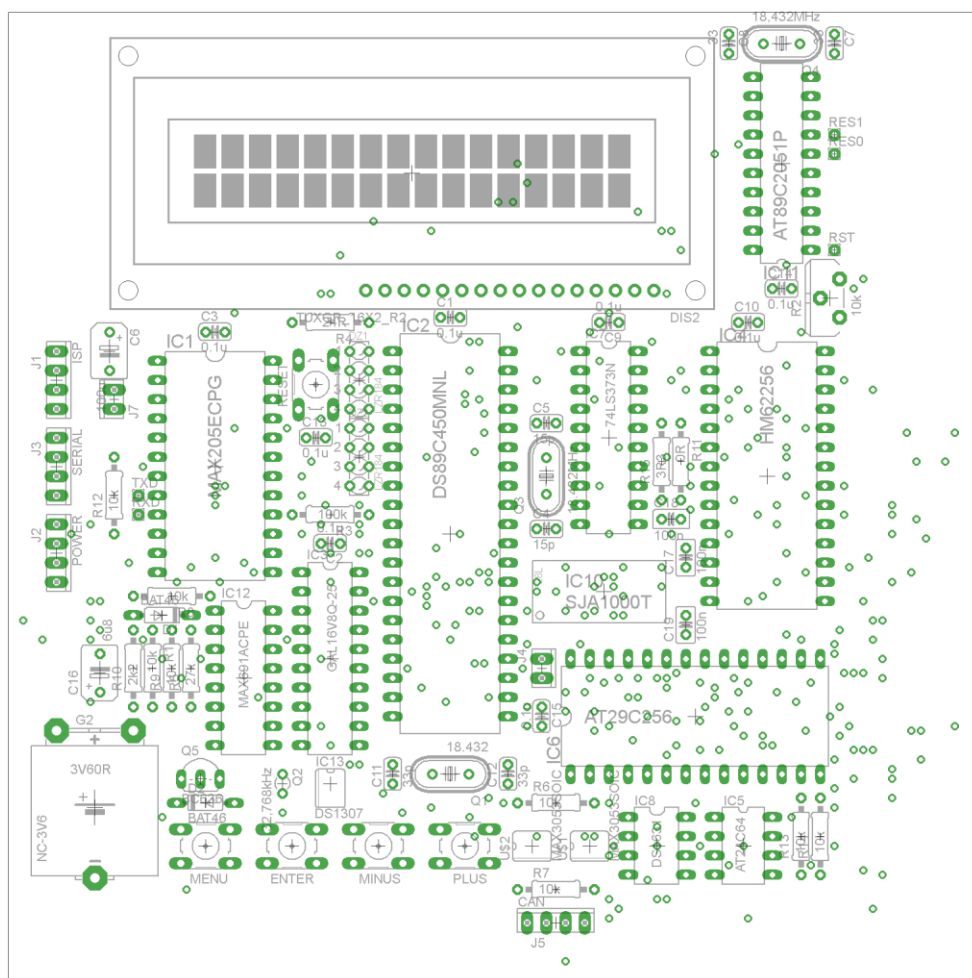


¹²⁶ Komunikačné rutiny CAN s RTX51 Full 7.0 nájdete: <http://www.keil.com/download/list/c51.htm>

Obrázok 148, Doska plošného spoja univerzálneho modulu s podporou CAN



Obrázok 149, Rozmiestnenie súčiastok univerzálneho modulu s podporou CAN



15.32. Ethernet s DS80C400

Dnešné moderné procesory¹²⁷ už vo svojom vnútri obsahujú hardware-ovú podporu technológie Ethernet 802.3, ktorá umožňuje navrhnuť ľubovoľné zariadenie s možnosťou komunikácie pomocou protokolov IPv4, alebo IPv6 cez internet. Všetky funkcie Ethernet sú uložené v ROM pamäti procesora a architektúra procesora umožňuje až 32 súčasných TCP spojení cez sieť. Procesor obsahuje IEEE 802.3 MII¹²⁸ interface. Maximálna pracovná frekvencia je 75MHz. Maximálne je možné obslúžiť procesorom pomocou 24-bit adresy až 16MB pamäti RAM. Štyri DPTR registre umožňujú rýchly prístup k dátam uloženým v XRAM. Spoločne s integrovanou hardware-ovou matematickou jednotkou užívateľ získava výkonný procesor pre ľubovoľnú aplikačnú oblasť.

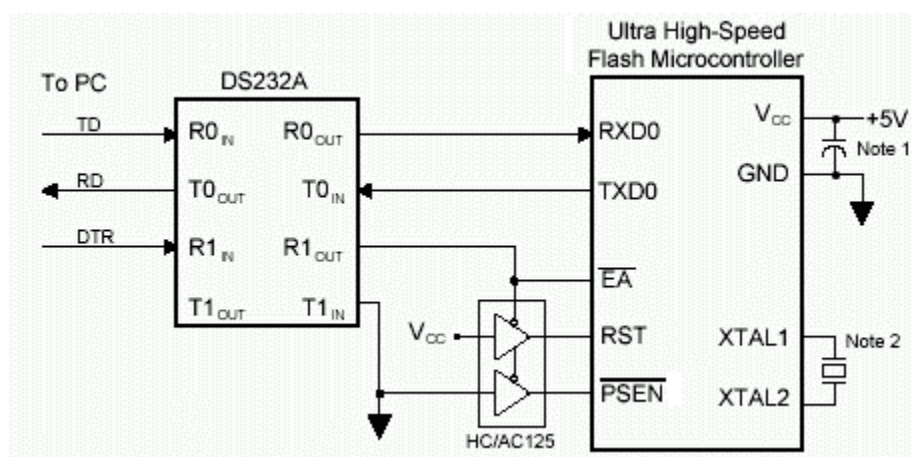
¹²⁷ <http://www.maximintegrated.com/en/products/digital/microcontrollers/DS80C400.html>

¹²⁸ <http://www.maximintegrated.com/en/products/interface/transceivers/78Q2123.html>

15.33. ISP rozhranie pre procesory

Problematika programovania súčiastok je pre každého pracovníka v oblasti mikroprocesorovej techniky dostatočne známa. Niečo iné je napísať program pre procesor a iné je naprogramovať skompilovaný program do procesora. Tu už je potrebné vo väčšine prípadov vlastniť špecializovaný programátor¹²⁹ ktorý je schopný zapísať program do hardware súčiastky. Pokiaľ je nutné programovať súčiastku priamo v aplikácii bez nutnosti jej bez toho aby sme ju odpojili tak nastáva problém ak súčiastka nepodporuje možnosť programovania cez ISP¹³⁰ rozhranie. Rozhranie ISP je pripojené cez externé súčiastky k PC pomocou sériového rozhrania RS232 a komunikuje pomocou software MTK¹³¹. Procesory ktoré sú programovateľné priamo zapojené v aplikácii cez ISP sa používajú napr. v PLC.

Obrázok 150, ISP pre procesory DS89C450



Obrázok 147 v kapitole 15.31 už obsahuje integrované obvodové zapojenie súčiastok ISP rozhrania pre programovanie procesora DS89C450 v zapojení. Prevodník úrovni z CMOS na RS232 je tvorený obvodom MAX205 a 74AC125 je nahradený kvôli jednoduchosti a dostupnosti programovateľným obvodom GAL16V8¹³².

¹²⁹ www.elnec.sk

¹³⁰ <http://www.maximintegrated.com/app-notes/index.mvp/id/3262>

¹³¹ http://www.maximintegrated.com/products/microcontrollers/software/dev_tool_software/mtk/MTKbeta.zip

¹³² GAL16V8 sa už neodporúča v nových aplikáciách a nahrádza sa obvodom GAL20V10.

15.34. Práca s ukazovateľom v jazyku C

Jazyk C disponuje nesmierne výkonným systémom typizovaných a netylizovaných ukazovateľov, ktorý sa nenachádza v iných programovacích jazykoch napr. Pascal, Basic Fortran a iné. Príklady¹³³ ktoré sú uvedené v obrázku tejto kapitoly uvádzajú jednotlivé použitia daných pointer-ov v príslušných konštrukčných volaniach funkcií a procedúr bez parametra alebo s parametrom.

Obrázok 151, Pointer v jazyku C

```
1  #include <reg52.h>
2  #include <stdio.h>
3  #include <stdlib.h>
4
5  int Counter ;
6
7  void Int_EX0 (void) interrupt 0 using 1
8  {
9      ++Counter ;
10 }
11
12 void Start (unsigned int i)
13 {
14     i++;
15 }
16
17 long Xyz (int i, int j, int k)
18 {
19     return (i+j+k);
20 }
21
22 void Func_caller (long (code *fptr) (unsigned int))
23 {
24     unsigned char i;
25     for (i=0; i<3; i++)
26     {
27         (*fptr)(i);    //Volanie funkcie Start(i) s odovzdaním hodnoty ...
28     }
29 }
30
31 void (*function_ptr1) (void); //Ukazovateľ na funkciu bez parametra ...
32
33 void (*function_ptr2) (int i, char j, char ch); //Ukazovateľ na funkciu s parametrom, v
APNT_129 je jeden parameter long ktorý sa nedokáže odovzdať cez parametre ...
34
35 void function_ptr (int i, char j, char ch)
36 {
37     i=j+ch;
38 }
39
40 long x;
41
42 void main (void)
43 {
44     while (1)
45     {
46         function_ptr1 = Int_EX0 ;
47         function_ptr2 = Xyz (0x1234, 0x1234abcd, 'a');
48         (*function_ptr) (0x1234, 'a', 'b');
49         //*****
50         Func_caller (Start); //Volanie funkcie Start(i) s odovzdaním hodnoty ...
51         ((void (code *) (void)) 0x0003) (); //Volanie funkcie alebo procedury na adrese 0x0003 ...
52         x=((long (code *) (int, int, int)) &Xyz) (1,2,3); //Volanie funkcie na adrese &Xyz s 3x
parametrami a navratom long ...
53         // x=((long (code *) (int, int, int)) 0x8000) (1,2,3); //Volanie funkcie na adrese 0x8000 s 3x
parametrami a navratom long ...
54     }
55 }
```

¹³³ Podrobnejšie informácie o pointer-och je možné získať na http://www.keil.com/appnotes/docs/apnt_129.asp

16 Programovanie aplikácií s využitím RTX51 Full

16.1. Problematika práce s RTX51 Full

Oproti predchádzajúcemu operačnému systému RTX51 Tiny je verzia Full komplexnejším a výkonnejším operačným systémom. Jadro operačného systému je napísané tak, aby umožňovalo programátorovi definovať samostatne fungujúce úlohy s reakciou na požadovaný typ udalosti a zaručeným časom odozvy t.j. odpovede.

Súčasne je RTX51 Full schopný riadiť maximálne 19 úloh, pričom 16 úloh môže pracovať s nižšou úrovňou priority, a tri úlohy je možno deklarovať ako **fast-tasks** s dobou reakcie pod 50 cyklov mikroprocesora. Prepnutie štandardnej úlohy je závislé na polohe a veľkosti zásobníka môže sa pohybovať v rozsahu 180 až 700 cyklov mikroprocesora. Naproti tomu prepnutie úlohy s úrovňou priority 3 (*fast task*) sa pohybuje v rozsahu 70 až 100 cyklov. Operačný systém RTX51 Full používa pre svoju činnosť len externú pamäť dát v ktorej si uchováva väčšinu informácií o task-och, informácia o polohe a veľkosti zásobníka.

Jedným obmedzením operačného systému RTX51 Full 5.0 z hľadiska programátora je to, že oblasť pamäti programu v rozsahu 0003H až 0080H kde sa štandardne nachádzajú obslužné programy vektorov prerušenia využíva RTX51 pre svoje potreby. Z vyššie uvedeného vyplýva, že programátor je úplne odkázaný na obsluhu prerušenia pomocou štandardných funkcií *os_attach_interrupt(number)* a *os_detach_interrupt(number)*. Až od verzie RTX51 Full 7.0 je možné vytvárať vlastné obslužné rutiny prerušenia. Tieto obslužné rutiny nesmú ale kolidovať s štandardnými prerušovacími (*interrupt*) funkciami uloženými na absolútnej adrese. Lepšiu predstavu o vektorech prerušenia používaných pri RTX51 nám poskytne súbor **RTXCONF.A51** spolu s **RTXSETUP.INC**. Súbor **RTX51.LIB** by mal byť linkerom pripojený až nakoniec. Pri tomto postupe sa vyhneme zbytočným chybovým hláseniam o kolízii pamäťových priestorov ktoré sú zdieľané vektormi prerušenia.

16.2. Použitie modifikovaných architektúr kompatibilných s 8051 s RTX51 Full

Pri mikroprocesore DS87C550¹³⁴ je potrebné pozmeniť časti programu **RTXCONF.A51**, pretože vektory prerušenia sú zmenené oproti štandardnej architektúre 8051.

Aj napriek tomuto nedostatku aplikácie vytvorené pomocou RTX51 sa vyznačujú vynikajúcimi vlastnosťami: variabilnými spôsobmi reakcií, pevne definovanými odozvami na prerušenie, vynikajúcimi schopnosťami vnútro-taskovej t.j. vnútro-úlohovej komunikácie,

¹³⁴ <http://pdfserv.maximintegrated.com/en/ds/DS87C550.pdf>

odolnosťou voči poruchám použitím separátnych stack-tasks oblastí, možnosť implementácie v rôznych typoch mikroprocesorov. Architektúra mikroprocesora DS87C550¹³⁵ približne 3x rýchlejšia ako konvenčné architektúry 8051 a prevod AD prevodníka je cca. 16,6μs oproti 50μs u 80C552.

Z bohatej praxe môžeme povedať, že na správnu činnosť RTX51 Full je nevyhnutné vyhradiť v externej pamäti XDATA¹³⁶ min. cca. 1kB pre jadro samotného operačného systému, min. 250 Byte +(použité premenné a deklarované dátové štruktúry) na každý task s prioritou 0 až 2. Pre task-y s prioritou 3 je použitá interná pamäť DATA, prípadne IDATA. Pri použití RTX51 Full sa snažíme, aby sme mali k dispozícii dostatočný priestor¹³⁷ aj pre zásobník procesora STACK, ktorý by mal mať minimálne aspoň 50 Byte, čo si môžeme skontrolovať vo výpise pri kompilácii programu. Keď si predstavíme, že samotný procesor 80c552 má k dispozícii len 256 Byte internej pamäti, musíme veľmi opatrne deklarovať a používať premenné ktoré sú uložené v DATA a IDATA.

¹³⁵ Procesor DS87C550 ktorý je derivátom 80C552 je už k dnešnému dňu 5.4.2014 firmou Maximintegrated nepodporovaný a nevyrábaný. Bol nahradený perspektívnejšími architektúrami.

¹³⁶ Pri použití operačného systému RTX51 Full sa deklarované premenné automaticky umiestňujú do externej pamäti dát XRAM.

¹³⁷ Dvojnásobne to platí pri použití reentrant-ných funkcií, kde je spotreba internej pamäte procesora priamo úmerná súčinu počtu volaní reentrantnej funkcie a veľkosti použitých premenných vo funkcii.

Obrázok 152, Implementácia do rtxconf.a51 o nepodporovaný processor DS87C550

c:\Keil\C51\Examples\rtx550\RTXCONF.A51

```
4052 ELSEIF (?RTX_CPU_TYPE = 44) ;Written by Ing. Eduard Jadron
4053
4054 ; Define the number and addresses of the interrupt enable registers.
4055 ; Set the not used registers to the same address as ?RTX_IE
4056 ;
4057 INT_EN_MASK_NUMBER EQU 2
4058 ?RTX_IE DATA 0A8H
4059 ?RTX_IEN1 DATA 0E8H
4060 ?RTX_IEN2 DATA 0A8H ; not used
4061
4062 ; Generate the interrupt entry points supported by the peripherals
4063 ; of the selected CPU type.
4064 ;
4065 IF (?RTX_SYSTEM_TIMER = 0)
4066 ; Do NOT include the Timer 0 Vector (INT-3)
4067 INT_ENTRY 0
4068 INT_ENTRY 1
4069 INT_ENTRY 2
4070 ; INT_ENTRY 3
4071 INT_ENTRY 4
4072 INT_ENTRY 5
4073 INT_ENTRY 6
4074 INT_ENTRY 7
4075 INT_ENTRY 8
4076 INT_ENTRY 9
4077 INT_ENTRY 10
4078 INT_ENTRY 11
4079 INT_ENTRY 12
4080 INT_ENTRY 13
4081 INT_ENTRY 14
4082 INT_ENTRY 15
4083 ELSEIF (?RTX_SYSTEM_TIMER = 1)
4084 ; Do NOT include the Timer 1 Vector (INT-10)
4085 INT_ENTRY 0
4086 INT_ENTRY 1
4087 INT_ENTRY 2
4088 INT_ENTRY 3
4089 INT_ENTRY 4
4090 INT_ENTRY 5
4091 INT_ENTRY 6
4092 INT_ENTRY 7
4093 INT_ENTRY 8
4094 INT_ENTRY 9
4095 ; INT_ENTRY 10
4096 INT_ENTRY 11
4097 INT_ENTRY 12
4098 INT_ENTRY 13
4099 INT_ENTRY 14
4100 INT_ENTRY 15
4101 ELSEIF (?RTX_SYSTEM_TIMER = 2)
4102 ; Do NOT include the Timer 2 Vector (INT-15)
4103 INT_ENTRY 0
4104 INT_ENTRY 1
4105 INT_ENTRY 2
4106 INT_ENTRY 3
4107 INT_ENTRY 4
4108 INT_ENTRY 5
4109 INT_ENTRY 6
4110 INT_ENTRY 7
4111 INT_ENTRY 8
4112 INT_ENTRY 9
4113 INT_ENTRY 10
4114 INT_ENTRY 11
4115 INT_ENTRY 12
4116 INT_ENTRY 13
4117 INT_ENTRY 14
4118 ; INT_ENTRY 15
4119 ENDIF
4120
4121 ; Select the Timer 2 configuration.
4122 ;
4123 ?RTX_TIMER2_TYPE EQU ?RTX_T2_STANDARD
4124
4125 ; The following table attaches the interrupt numbers (0..31) to the
```

c:\Keil\C51\Examples\rtx550\RTXCONF.A51

```
4126      ; corresponding bits in the interrupt enable masks of the specific
4127      ; processor.
4128      ; All three interrupt enable register contents must be defined
4129      ; for every interrupt number (even when the specific processor contains
4130      ; only one interrupt mask).
4131      ; Syntax: DB IE-content, IE1-content, IE2-content
4132      ;
4133      ?RTX?RTX_INT_TO_BIT_TABLE?RTXCONF  SEGMENT  CODE
4134      RSEG  ?RTX?RTX_INT_TO_BIT_TABLE?RTXCONF
4135
4136      ?RTX_INT_TO_BIT_TABLE_BASE :
4137          DB 01H, 00H, 00H      ; INT_0,  nINTT0 - INT_EX0
4138          DB 20H, 00H, 00H      ; INT_1,  SCON1 at P1
4139          DB 40H, 00H, 00H      ; INT_2,  A/D Analog Converter
4140          DB 02H, 00H, 00H      ; INT_3,  TF0 - Timer 0
4141          DB 00H, 01H, 00H      ; INT_4,  INT2/CF0
4142          DB 00H, 10H, 00H      ; INT_5,  CM0F
4143          DB 00H, 80H, 00H      ; INT_6,  PFI Power Fail Interrupt
4144          DB 04H, 00H, 00H      ; INT_7,  nINTT1 - INT_EX1
4145          DB 00H, 02H, 00H      ; INT_8,  INT3/CF1
4146          DB 00H, 20H, 00H      ; INT_9,  CM1F
4147          DB 08H, 00H, 00H      ; INT_10, TF1 - Timer 1
4148          DB 00H, 04H, 00H      ; INT_11, INT4/CF2
4149          DB 00H, 40H, 00H      ; INT_12, CM2F
4150          DB 10H, 00H, 00H      ; INT_13, SCON0 at P0
4151          DB 00H, 04H, 00H      ; INT_14, INT5/CF3
4152          DB 00H, 80H, 00H      ; INT_15, TF2 - Timer 2
4153
4154      ; Define the greatest supported interrupt number
4155      ;
4156      ?RTX_MAX_INT_NBR  EQU  15
4157
4158      ; Enter Idle Mode Macro
4159      ; Used whenever entering idle state. Peripherals stay active.
4160      ; Leaved after interrupt.
4161      ;
4162      PCON  DATA  87H
4163      ENTER_IDLE MACRO
4164          ORL  PCON, #01H      ; Set idle mode
4165          ENDM
4166
4167      ; Define the register addresses to set the MSB of the page address.
4168      ;
4169      ?RTX_PAGE_OUT_REG DATA 0A0H      ; write directly to Port 2
4170
4171      ELSE
```


Obrázok 153, Modifikácia súboru RTXCONF.A51 pre procesor DS87C550

c:\Keil\C51\Examples\rtx550\RTXSETUP.INC

```
60 ; 1. CPU TYPE
61 ; =====
62 ;
63 ; CPU_TYPE stands for the desired microprocessor type and can be
64 ; selected from the following table:
65 ;
66 ; Manufacturer      Model      CPU_TYPE
67 ; -----
68 ; Acer Lab          M6759      41
69 ; Aeroflex UPMC      UT69RH051  23
70 ; Atmel              87F51      1
71 ; Atmel              87F51RC     2
72 ; Atmel              87F52      2
73 ; Atmel              89C51      1
74 ; Atmel              89C52      2
75 ; Atmel              89C55      2
76 ; Atmel              89F51      1
77 ; Atmel              89F52      2
78 ; Atmel              89LS53     2
79 ; Atmel              89LS8252    2
80 ; Atmel              89LV52     2
81 ; Atmel              89LV55     2
82 ; Atmel              89S53      2
83 ; Atmel              89S8252    2
84 ; Atmel Wireless    80C31      1
85 ; Atmel Wireless    80C31X2     1
86 ; Atmel Wireless    80C32      2
87 ; Atmel Wireless    80C51      1
88 ; Atmel Wireless    80C51RA2    5
89 ; Atmel Wireless    80C51RD2    5
90 ; Atmel Wireless    80C51U2     5
91 ; Atmel Wireless    80C52X2     2
92 ; Atmel Wireless    80C54X2     2
93 ; Atmel Wireless    80C58X2     2
94 ; Atmel Wireless    83 /87C51RB2  5
95 ; Atmel Wireless    83 /87C51RC2  5
96 ; Atmel Wireless    83 /87C51RD2  5
97 ; Atmel Wireless    83 /87C51U2     5
98 ; Atmel Wireless    83 /87C52X2     2
99 ; Atmel Wireless    87C51      1
100 ; Atmel Wireless    T87C5112    29
101 ; Atmel Wireless    T89C51CC01   42
102 ; Atmel Wireless    T89C51RD2    5
103 ; Cygnal              C8051F005 /6/7  39 [* 2]
104 ; Cygnal              C8051F015 /16 /17  39 [* 2]
105 ; Cygnal              C8051F020 /21 /22 /23  43 [* 2]
106 ; Cygnal              C8051F206     40 [* 2]
107 ; Cygnal              C8051F226     40 [* 2]
108 ; Cygnal              C8051F236     40 [* 2]
109 ; Dallas              DS80C310     33
110 ; Dallas              DS80C320     16 [* 3]
111 ; Dallas              DS80C323     16 [* 3]
112 ; Dallas              DS80C530     17 [* 3]
113 ; Dallas              DS87C520 /DS83C520  16 [* 3]
114 ; Dallas              DS87C530     17 [* 3]
115 ; Dallas              DS89C420     16 [* 3]
116 ; Dallas              DS87C550     44 [* 2] [* 3??] Written by Ing. Eduard Jadron
117 ; Hynix (Hyundai)     GMS90C32     2
118 ; Hynix (Hyundai)     GMS90C320    2
119 ; Hynix (Hyundai)     GMS90C51     1
120 ; Hynix (Hyundai)     GMS90C52     2
121 ; Hynix (Hyundai)     GMS90C54     2
122 ; Hynix (Hyundai)     GMS90C56     2
123 ; Hynix (Hyundai)     GMS90C58     2
124 ; Hynix (Hyundai)     GMS90L32     2
125 ; Hynix (Hyundai)     GMS90L320    2
126 ; Hynix (Hyundai)     GMS90L51     1
127 ; Hynix (Hyundai)     GMS90L52     2
128 ; Hynix (Hyundai)     GMS90L54     2
129 ; Hynix (Hyundai)     GMS90L58     2
```

17 Príklady programov s RTX51 Full

17.1. Absolútny a prírastkový diskretný regulátor PSD

Príklad: Pomocou programovacieho jazyka C51 a RTX51 vytvorte program, ktorý bude vykonávať funkciu regulátora s prírastkovým algoritmom PSD pre reguláciu pohonu jednosmerného motora s cudzím budením s parametrami: $T_i=0.45s$, $T_d=0.02s$, $\Delta T=20ms$, $K=0.22$, $U=230V$, $I_A=10A$. Žiadaná hodnota prúdu I_z bude vzorkovaná s periódou minimálne $\Delta T=20ms$ na porte P5.0 a skutočná hodnota I_s na porte P5.1 ako 10 bitová analógová hodnota. Výstupná akčná veličina y_N na výstupe regulátora bude zapisovaná do registra PWM0.

Rozbor úlohy: Klasický tvar **PID** algoritmu regulácie je možné vyjadriť vzt'ahom:

$$y(t) = K \cdot \left[e(t) + \frac{1}{T_i} \int e(t) dt + T_d \frac{de(t)}{dt} \right] \quad (7.)$$

Absolútny tvar algoritmu regulátora **PSD** je možné zapísať pomocou vzt'ahu:

$$y_N = K \cdot \left[e_N + \frac{1}{T_i} \sum_{i=1}^N \frac{e_i + e_{i-1}}{2} \Delta T + \frac{T_d \cdot (e_N - e_{N-1})}{\Delta T} \right] \quad (8.)$$

Prírastkový algoritmus regulátora **PSD** je možné zapísať po úprave vyššie uvedeného vzt'ahu¹³⁸:

$$y_N = y_{N-1} + \left[K \cdot \left(1 + \frac{\Delta T}{2T_i} + \frac{T_d}{\Delta T} \right) \cdot e_N - K \cdot \left(1 - \frac{\Delta T}{2T_i} + \frac{2T_d}{\Delta T} \right) \cdot e_{N-1} + K \cdot \left(\frac{T_d}{\Delta T} \right) \cdot e_{N-2} \right] \quad (9.)$$

t.j. po zjednodušení

$$y_N = y_{N-1} + A \cdot e_N - B \cdot e_{N-1} + C \cdot e_{N-2} \quad (10.)$$

kde **A, B, C** sú konštanty a e_N je odchýlka žiadanej a skutočnej regulovanej veličiny

$$A = K \cdot \left(1 + \frac{\Delta T}{2T_i} + \frac{T_d}{\Delta T} \right) \quad (11.)$$

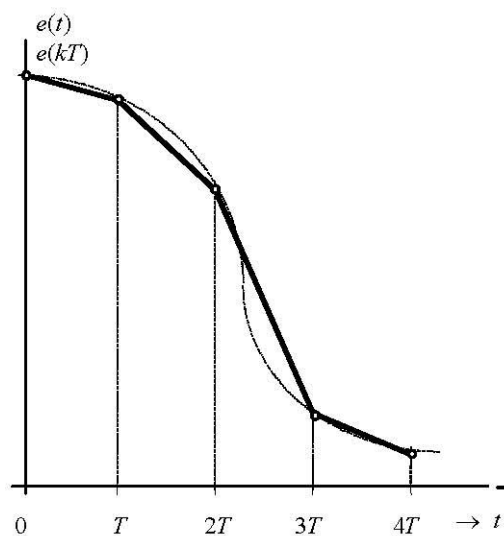
$$B = K \cdot \left(1 - \frac{\Delta T}{2T_i} + \frac{2T_d}{\Delta T} \right) \quad (12.)$$

$$C = K \cdot \left(\frac{T_d}{\Delta T} \right) \quad (13.)$$

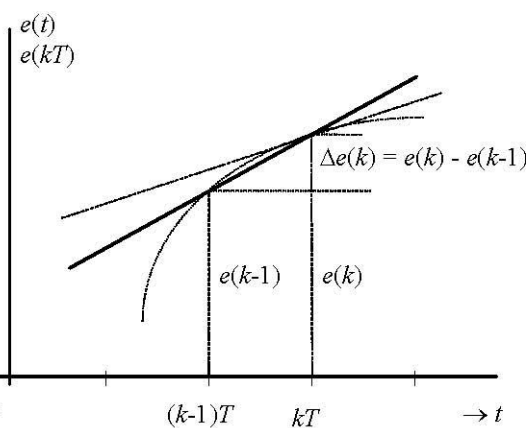
¹³⁸ Pri použití celočíselnej aritmetiky je tento vzt'ah vypočítaný pri 18.432MHz za cca. 400 μs , pri reálnej aritmetike je tento výpočet ukončený za 1760 μs . S výberom hodnôt z tabuľky je tento výpočet ukončený asi za 150 μs .

Obrázok 154, Náhrada integrálu sumou lichobežníkov

a. sprava

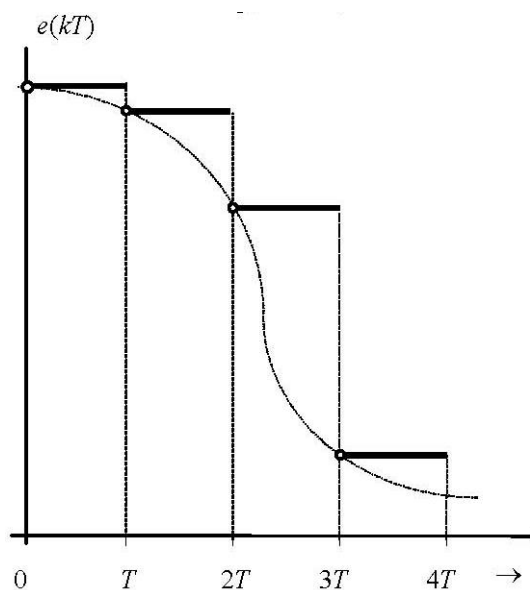


b. zľava

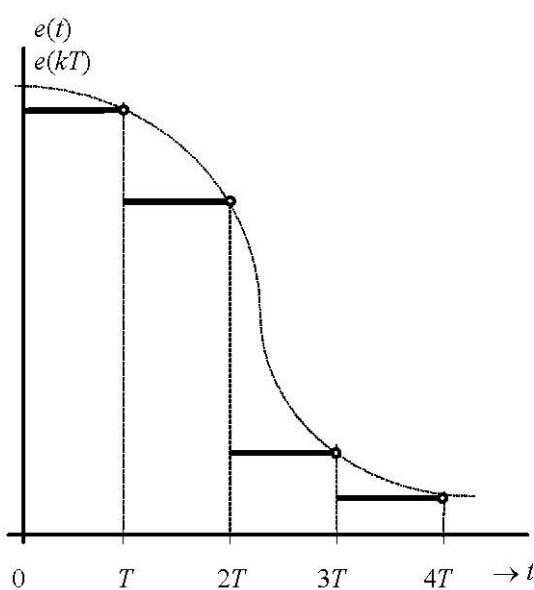


Obrázok 155, Náhrada integrálu sumou obdĺžnikov

a. sprava



b. zľava



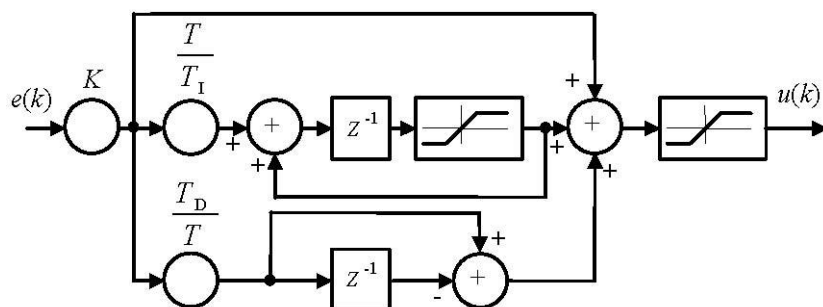
Pri použití náhrady integrálu obdĺžnikmi dostaneme výsledný vzťah:

$$y_N = y_{N-1} + \left[K \cdot \left(1 + \frac{T_d}{\Delta T} \right) \cdot e_N - K \cdot \left(1 - \frac{\Delta T}{T_i} + \frac{2T_d}{\Delta T} \right) \cdot e_{N-1} + K \cdot \left(\frac{T_d}{\Delta T} \right) \cdot e_{N-2} \right] \quad (14.)$$

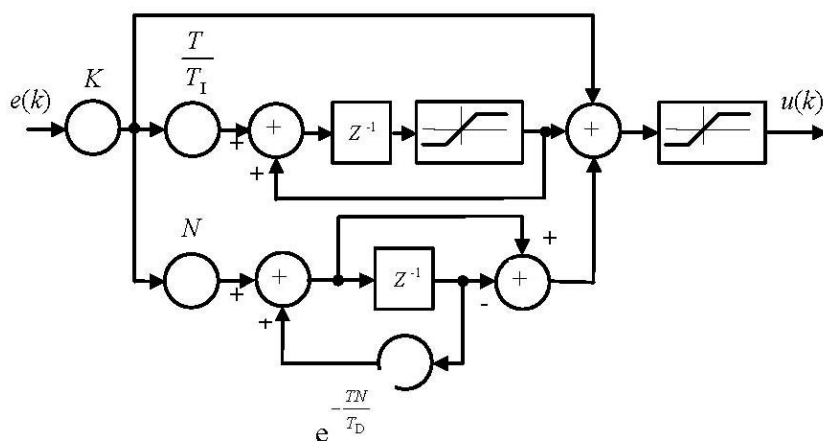
Pri použití náhrady integrálu lichobežníkmi dostaneme výsledný vzťah:

$$y_N = y_{N-1} + \left[K \cdot \left(1 + \frac{\Delta T}{2T_i} + \frac{T_d}{\Delta T} \right) \cdot e_N - K \cdot \left(1 - \frac{\Delta T}{2T_i} + \frac{2T_d}{\Delta T} \right) \cdot e_{N-1} + K \cdot \left(\frac{T_d}{\Delta T} \right) \cdot e_{N-2} \right] \quad (15.)$$

Obrázok 156, Stavový diagram polohového PSD regulátora



Obrázok 157, Stavový diagram polohového PSD regulátora s filtráciou derivačnej zložky



Zápis programu:

```
#include <c:\keil\c51\inc\reg552.h>
#include <c:\keil\c51\inc\stdio.h>
#include <c:\keil\c51\inc\ctype.h>
#include <c:\keil\c51\inc\string.h>
#include <c:\keil\c51\inc\stdlib.h>
#include <c:\keil\c51\inc\rtx51.h>
#include <c:\keil\c51\inc\absacc.h>
#include <c:\keil\c51\inc\math.h>

/* standard I/O .h-file */
/* character functions */
/* string and memory functions */

#define INIT 0
#define COMMAND 1
#define ANALOG 2
#define CLOCK 3

xdata float K,Td,Ti,dT,U,Iz,Is;
xdata float a,b,c;
xdata float En[2];
xdata float Yn[2];

#define CHANNELS 2
idata unsigned int Result_ADC[CHANNELS];

void Command(void) _task_ COMMAND _priority_ 1
{
    Iz=10;
    U=230.0;
    K=0.22;
    Ti=0.45;
    Td=0.2;
    dT=0.02;
    a=K*(1.0+(dT/(2*Ti)))+(Td/dT);
    b=K*(1.0-(dT/(2*Ti)))+(2*Td/dT);
    c=K*(Td/dT);
    Is=9.99;
    while(1)
    {
        os_wait(K_TMO,2,0);
        os_send_signal(ANALOG);
        Is=(float)Result_ADC[0];
        Iz=(float)Result_ADC[1];
        En[0]=Iz-Is;
        Yn[0]=Yn[1]+a*En[0]-b*En[1]+c*En[2];
        if(Yn[0]>255) Yn=255;
        if(Yn[0]<0) Yn=0;
        PWM0=(unsigned char)Yn;
        Yn[1]=Yn[0];
        En[2]=En[1];
        En[1]=En[0];
    }
}

xdata unsigned int Ticks;

void Clock(void) _task_ CLOCK _priority_ 1
{
    while(1)
    {
        os_wait(K_TMO,100,0);
        Ticks++;
    }
}
```

```
void Init(void) _task_ INIT _priority_ 0
{
    while(1)
    {
        os_set_slice(10000);
        os_create_task(COMMAND);
        os_create_task(CLOCK);
        os_create_task(ANALOG);
        os_delete_task(INIT);
    }
}

#pragma REGISTERBANK (3)

void Analog(void) _task_ ANALOG _priority_ 3
{
    #define ADCS 0x08
    #define ADCI 0x10
    #define ADEX 0x20
    unsigned char ADC_Channel;
    while(1)
    {
        os_wait(K_SIG,0xFF,0);
        for(ADC_Channel=0x00; ADC_Channel<CHANNELS; ADC_Channel++)
        {
            ADCON=ADC_Channel;

            ADCON|=ADEX+ADCS; //Vyberiem kanal ADC0..ADC7 pre prevod
            while((ADCON&ADCI)==0x00); //Externe spustanie prevodu+STDAC nast. ADEX=1
            ADCON^=ADCI; //Pockam, pokiaľ neskonci prevod
            Result_ADC[ADC_Channel]=(256*ADCH+(ADCON&0xC0)>>6); //Vysledok prevodu rusim ADCI=0
        }
    }
    os_detach_interrupt(10);
}

#pragma REGISTERBANK (0)

void main(void)
{
    os_start_system(INIT);
}
```


17.2. 8-bitový PSD regulátor

c:\Dropbox\Záloha\C51\PSD8bit\PSD8.c

```
1  #include <reg51.h>
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <string.h>
5
6  signed char  KA, KB, KC, En, En1, En2, Yn, Yn1, Iz, Is;
7
8  void main (void)
9  {
10     KA=1;
11     KB=2;
12     KC=3;
13     while (1)
14     {
15         Iz =P1;
16         Is =P3;
17         En =Iz - Is;
18         Yn=Yn1+( KA *En )-( KB *En1 )+( KC *En2 );
19         Yn1 =Yn;
20         En2 =En1;
21         En1 =En;
22     }
23 }
24
```

17.3. 32-bitový PSD regulátor

C:\Dropbox\Záloha\C51\PSD\PSD.C

```
130 void Command(void) _task_ COMMAND _priority_ 2
131 {
132     Iz=10;
133     Is=9.99;
134     U=230.0;
135     K=0.22;
136     Ti=0.45;
137     Td=0.2;
138     dT=0.02;
139     Ka=K*(1.0+(dT/(2*Ti)))+(Td/dT);
140     Kb=K*(1.0-(dT/(2*Ti)))+(2*Td/dT);
141     Kc=K*(Td/dT);
142     while(1)
143     {
144         os_wait(K_SIG,0xFF,NULL);
145         Is=(float)Result_ADC[0];
146         Iz=(float)Result_ADC[1];
147         En=Iz-Is;
148         Yn=Yn1+(Ka*En)-(Kb*En1)+(Kc*En2);
149         // if (Yn>255) Yn=255;
150         // if (Yn<0) Yn=0;
151         PWM0=(unsigned char) Yn;
152         Yn1=Yn;
153         En2=En1;
154         En1=En;
155     }
156 }
```

17.4. Jednosmerný impulzový menič v znižovacom zapojení STEP-DOWN

Impulzové meniče sú elektrické prístroje vyznačujúce sa vysokou účinnosťou a malým objemovým koeficientom. Pracovná frekvencia impulzových meničov sa nachádza v rozsahu jednotiek až desiatok kHz, ktorá siahá až do jednotiek MHz. Účinnosti jednotlivých impulzových meničov je závislá na obvodovej topológii meniča – obvodové zapojenie a môže byť v rozsahu od 60% do 99,5%. Návrh impulzového meniča je v porovnaní s lineárnym meničom omnoho komplikovanejší a náročnejší.

Teoretický rozbor impulzového meniča STEP-DOWN (BUCK).

Matematický opis je možné rozdeliť na dve časti:

Spínač S1 zopnutý počas doby t_1 , takže napätie U_L na indukčnosti vzrastá

$$U_L = L \frac{dI_1}{dt} \quad (16.)$$

Z vyššie uvedeného vyjadríme I_1

$$dI_1 = \frac{U_1 - U_2}{L} \cdot t_1 \quad (17.)$$

Spínač S1 rozopnutý počas doby t_2 , takže napätie U_L na indukčnosti klesá podľa

$$U_L = L \frac{dI_2}{dt} \quad (18.)$$

Z vyššie uvedeného vyjadríme I_2

$$-dI_2 = -\frac{U_2}{L} \cdot t_2 \quad (19.)$$

Prúdy I_1 a I_2 sú vďaka indukčnosti zhodné, takže môžeme písať

$$U_2 \cdot t_2 = \frac{U_1 - U_2}{t_1} \quad (20.)$$

Úpravou vzťahu predchádzajúceho vzťahu získame

$$U_2 = U_1 \cdot \frac{t_1}{t_1 + t_2} = U_1 \cdot \frac{t_1}{T} = U_1 \cdot \delta \quad (21.)$$

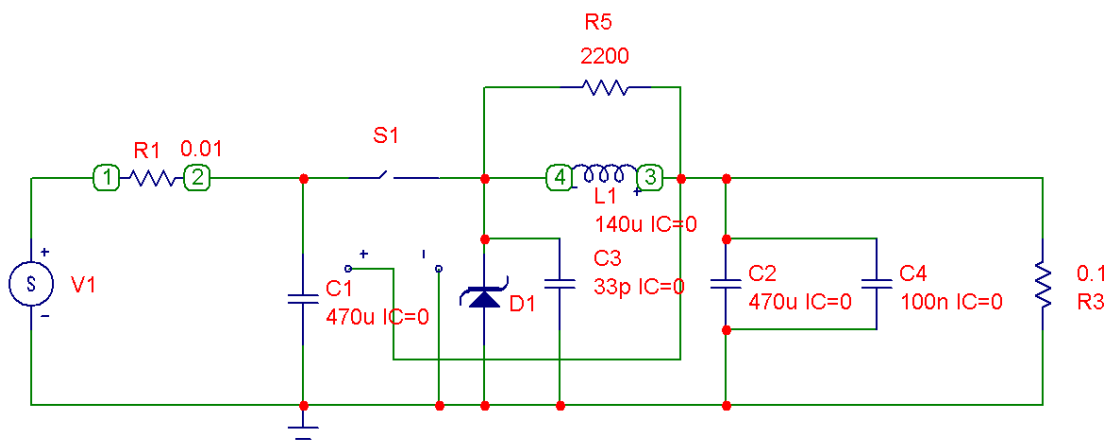
Pokiaľ budeme uvažovať účinnosť $\eta = 100\%$ môžeme písať

$$U_1 \cdot I_1 = U_2 \cdot I_2 \quad (22.)$$

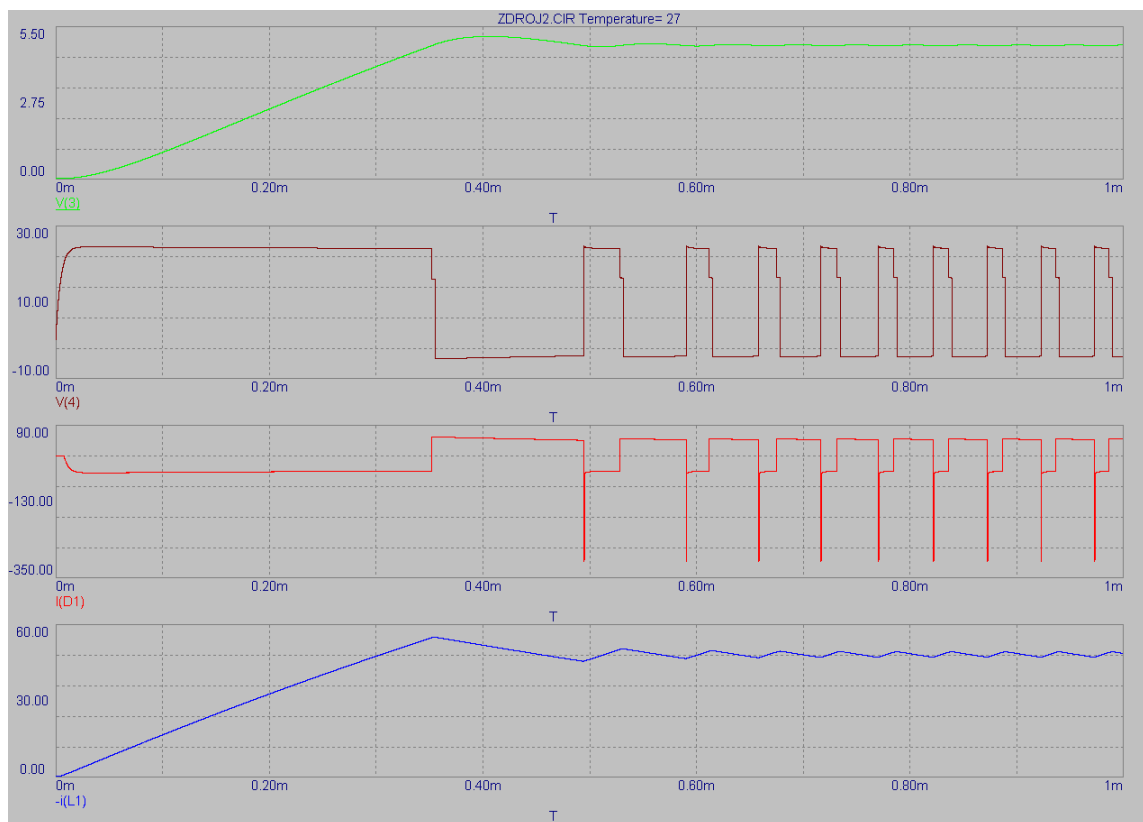
Z čoho vyjadríme výstupný prúd I_2

$$I_2 = I_1 \cdot \frac{U_1}{U_2} \quad (23.)$$

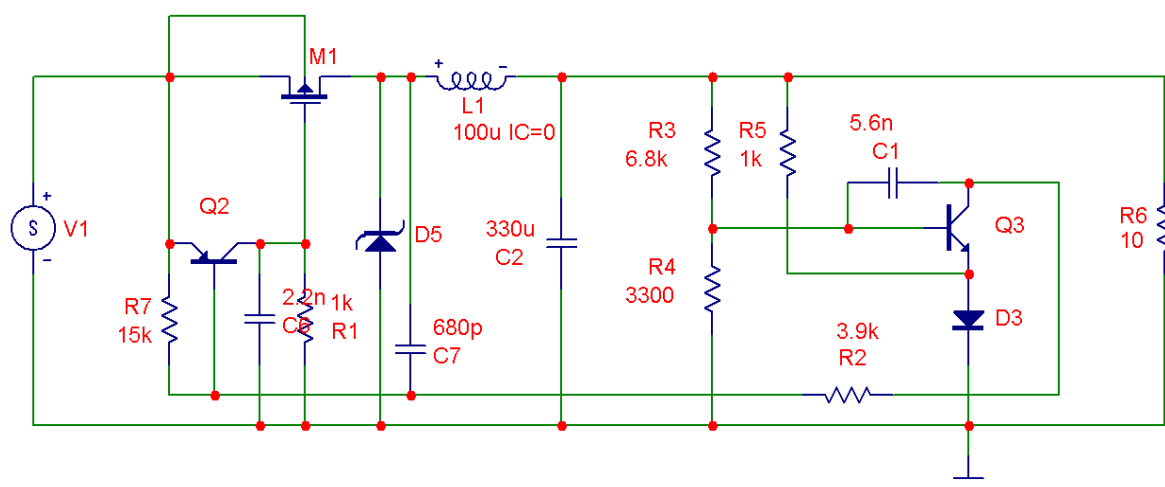
Obrázok 158, Simulačný model impulzového meniča



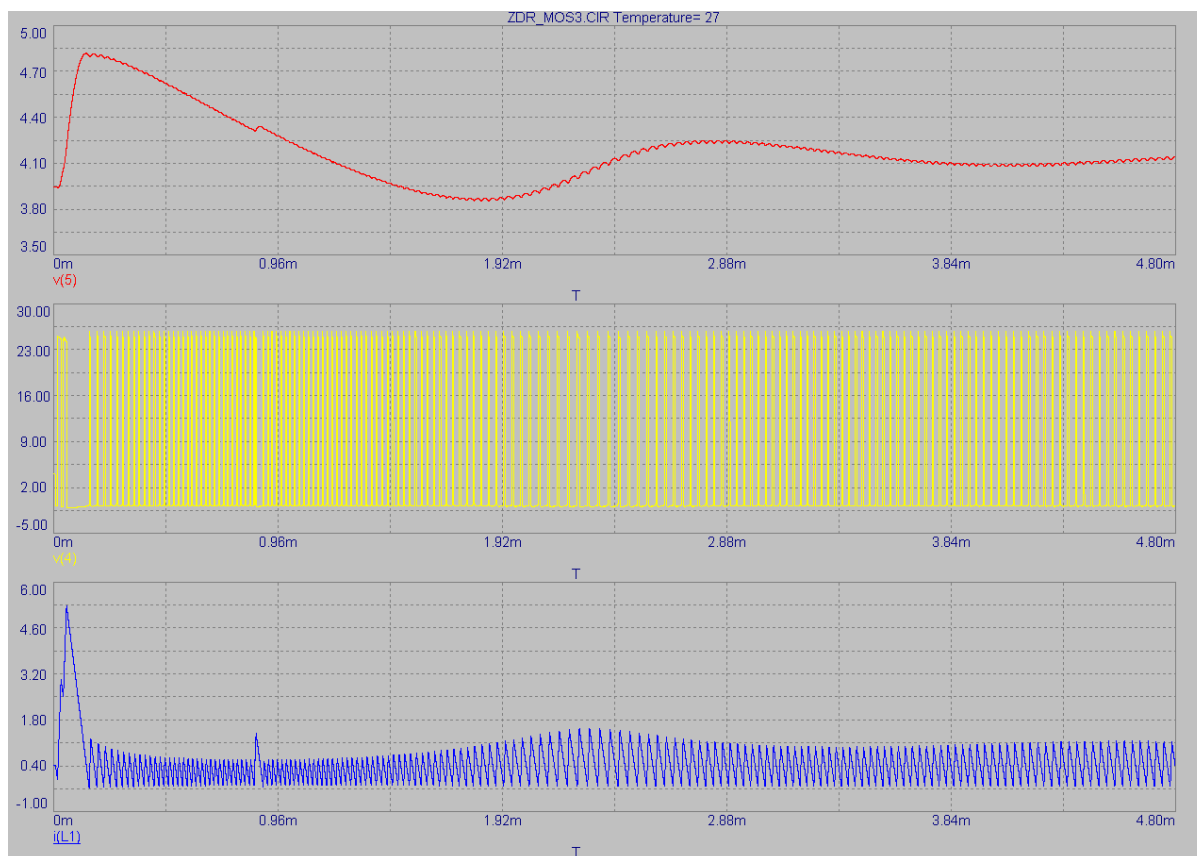
Obrázok 159, Simulačné výsledky modelu impulzového meniča



Obrázok 160, Reálne zapojenie impulzového meniča s reguláciou výstupného napätia



Obrázok 161, Simulačné výsledky reálneho zapojenia impulzového meniča

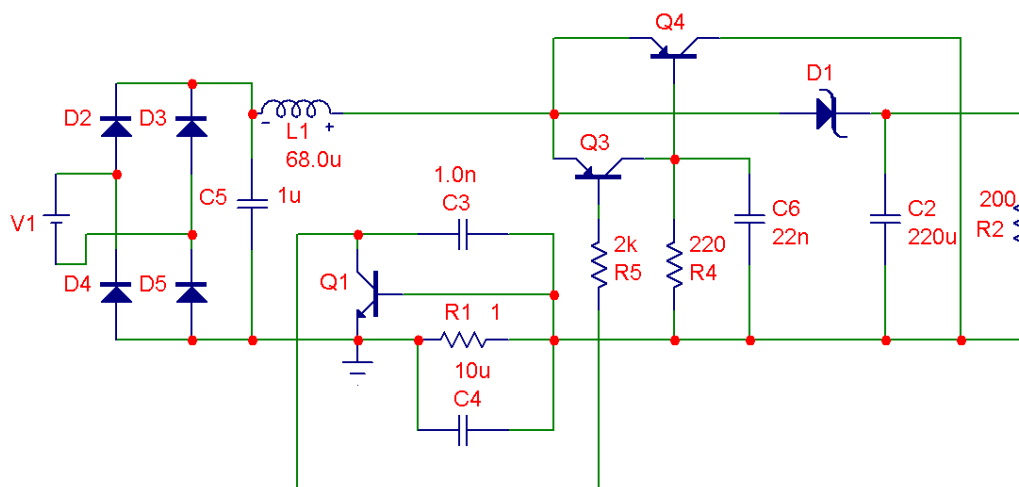


Poznámka autora: Vyššie uvedené zapojenie impulzového zdroja bolo použité ako náhrada lineárneho zdroja v čiernobielych TVP a nabíjaciach adaptéroch pre mobilné telefóny. Účinnosť tohto zdroja je $\eta = 86\%$ pri pracovnej frekvencii $f \approx 100\text{kHz}$. S výhodou je možné použiť tento typ zdroja pre napájanie spätného osvetlenia LCD panelov ktoré sú použité v aplikáciách s mikroprocesormi, alebo na napájanie pomocných obvodov.

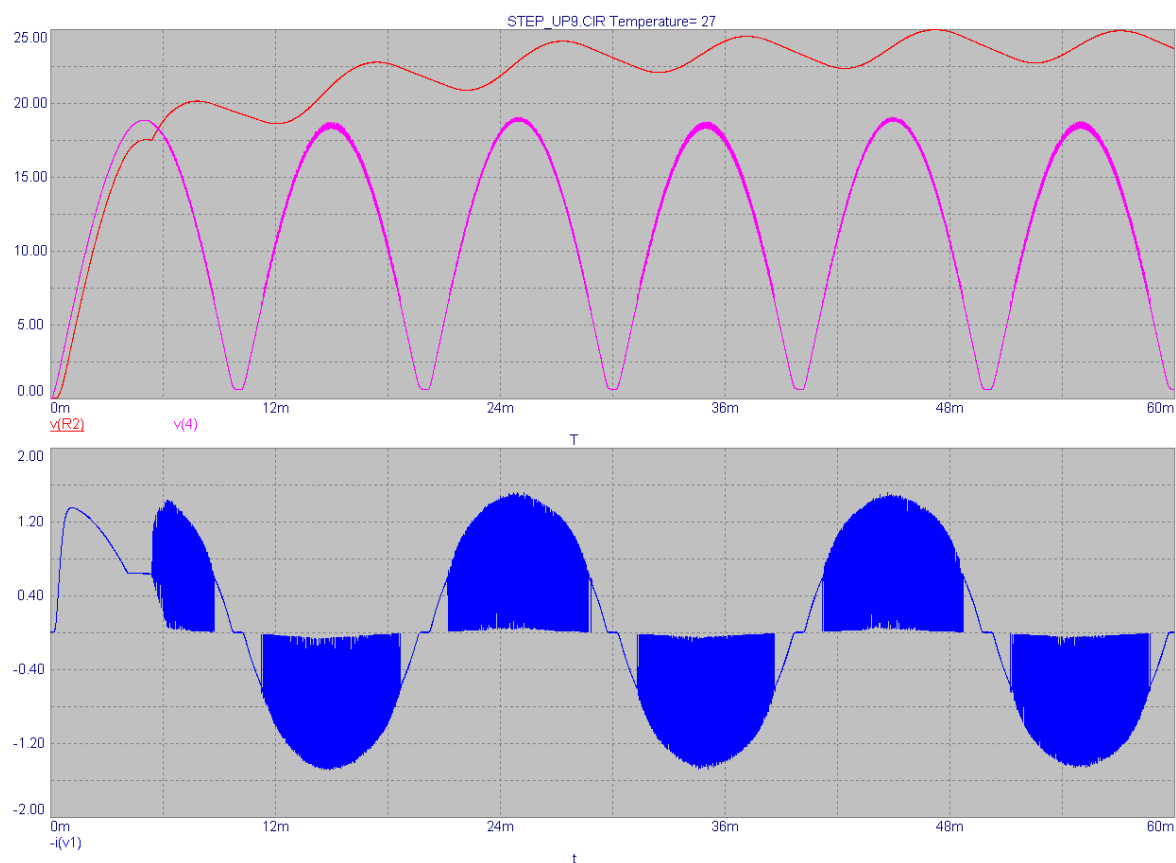
17.5. Jednosmerný impulzový menič v zvyšovacom zapojení STEP-UP s PFC

Obrázok 162 predstavuje zapojenie jednoduchého impulzového meniča s PFC korekciou odoberaného vstupného prúdu meniča. V tomto zapojení je schopný menič odoberať prúd ktorého tvar po aproximácii môžeme porovnávať so sínusovým priebehom. Tento menič je používaný vo väčšine moderných prístrojov, pretože umožňuje vykonávať aj činnosť usmerňovača a aj funkciu zvyšovacieho meniča, ktorý umožňuje odoberať výkon zo striedavej siete s účinníkom $\eta \div 100\%$.

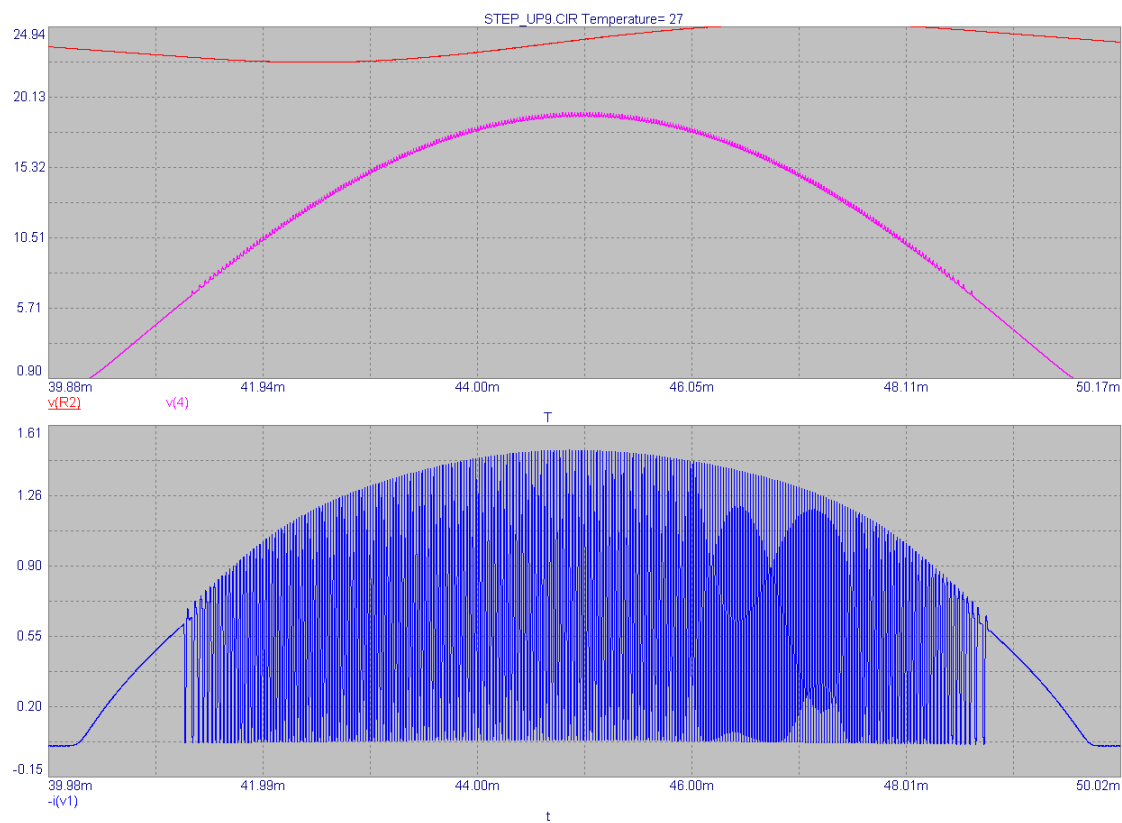
Obrázok 162, Reálne zapojenie impulzového meniča s PFC – Power Factor Correction



Obrázok 163, Simulačné výsledky reálneho zapojenia impulzového meniča s PFC



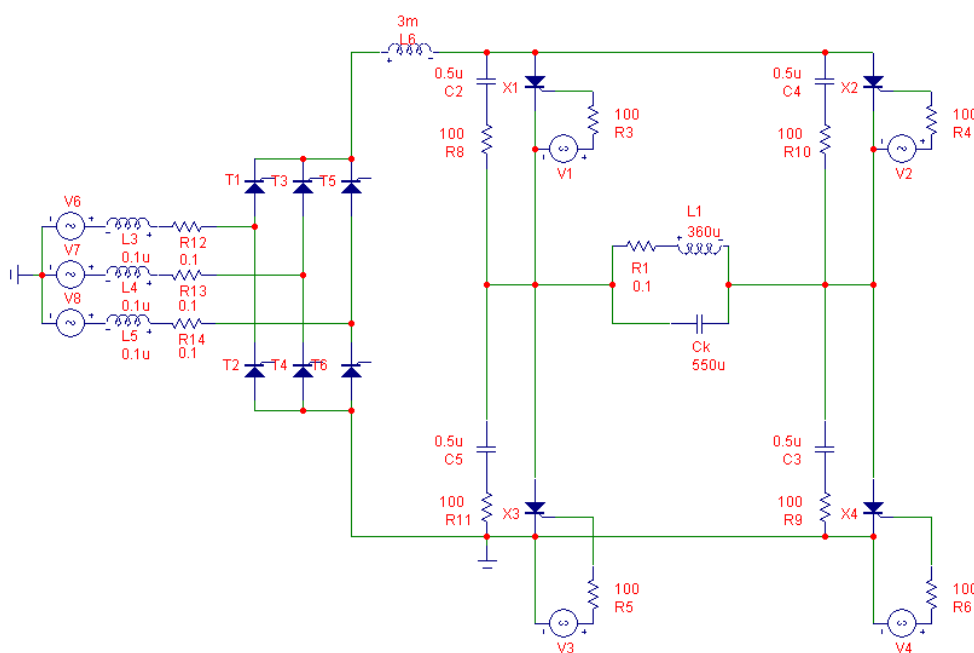
Obrázok 164, Detailné simulačné výsledky impulzového meniča s PFC



17.6. Diskrétné riadenie 3-fázového usmerňovača s tyristormi

Príklad: S využitím programovacieho jazyka C51 a RTX51 vytvorte diskretný regulátor pre mostový usmerňovač s tyristormi pre elektrické pohony s jednosmernými motormi, alebo indukčné pece. Daný regulátor musí byť schopný samostatne generovať všetky zapínacie impulzy pre jednotlivé polovodičové prvky bez prídavných obvodov. Regulačný algoritmus PSD použite z predchádzajúcej úlohy. Využite záchytné (CTH0..2, CTL0..2) a porovnávacie (CMH0..2, CML0..2) registre mikroprocesora 87C552 pre meranie dĺžky periódy siete a pre generovanie zapínacích impulzov s fázovým posuvom α v rozsahu 0° až 150° elektrických. Na výstupných svorkách portu P4.0 až P4.2 budú generované primárne zapínacie impulzy I_1, I_2, I_3 , pre tyristory usmerňovača. Sekundárne impulzy budú odvodené od primárnych impulzov a smeru otáčania jednotlivých fázových napätí siete. Zapínací impulz¹³⁹ pre tyristory bude mať dĺžku maximálne $t_{ON} = 150\mu s$. Signál smeru otáčania bude dostupný na P4.3 a bude vstupovať do PLD obvodu s označením GAL16V8B¹⁴⁰, ktorý bude generovať primárne a sekundárne impulzy pre jednotlivé tyristory v kladnej a zápornej skupine.

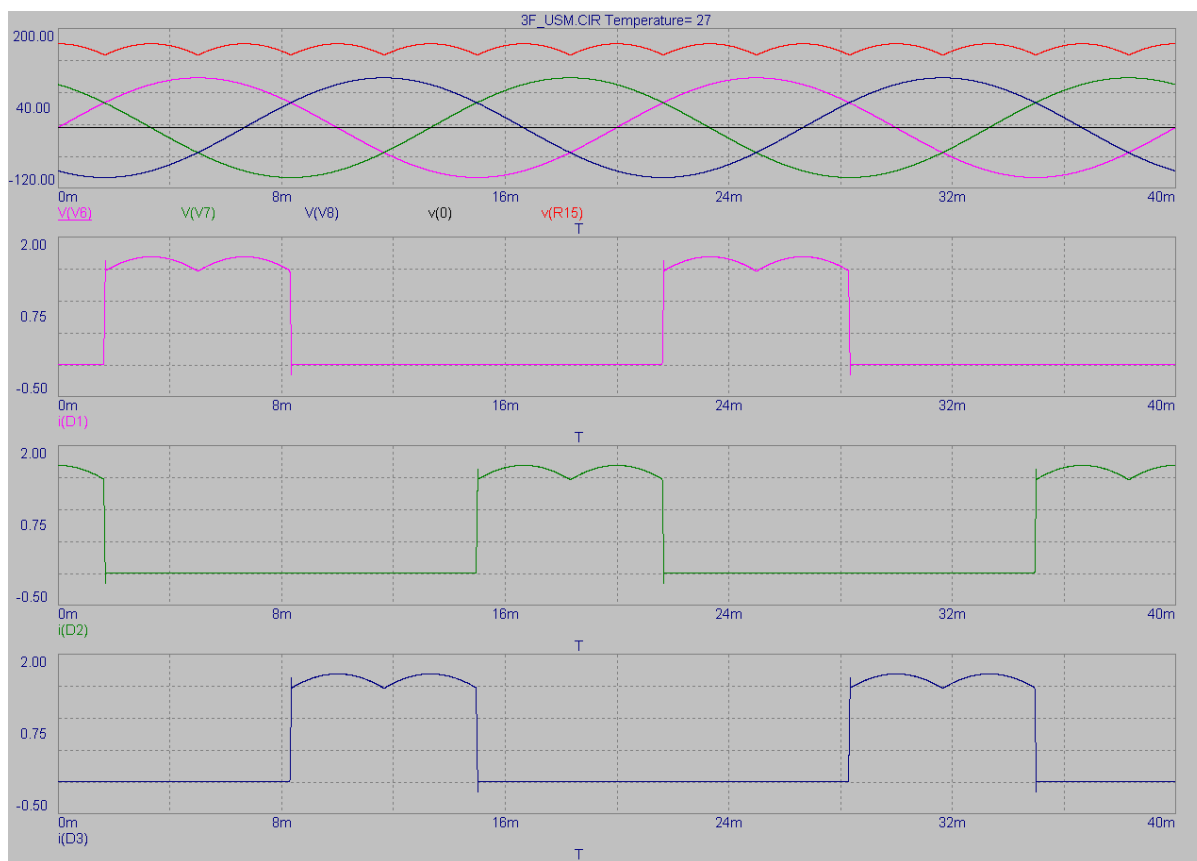
Obrázok 165, Mostový 3-fázový usmerňovač s napäťovým striadačom a rez. záťažou



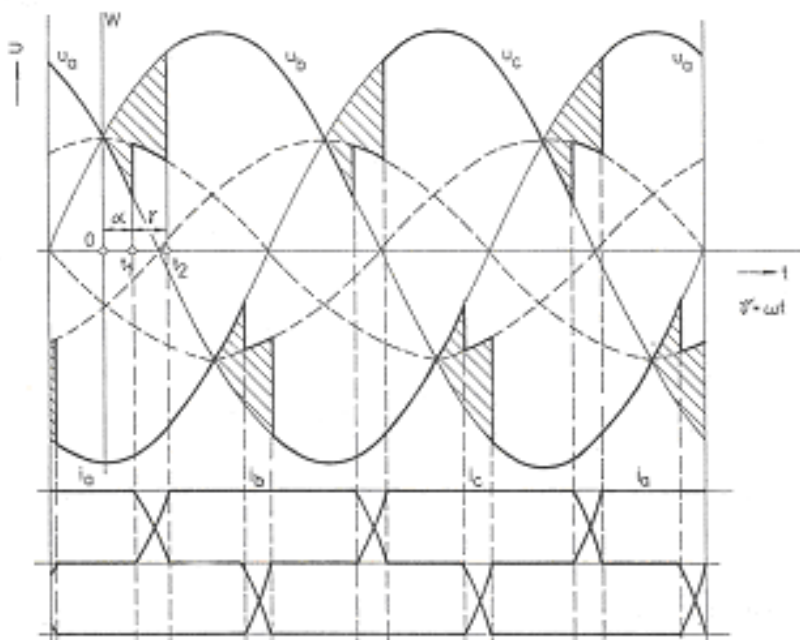
¹³⁹ Dĺžka zapínacieho impulzu privedená na radiacu elektródu pre tyristor usmerňovača TV960/12/400 musí byť minimálne 85 μs . Je to potrebné pre impulzový transformátor, aby vyrobil dostatočne sýty prúdový zapínací impulz.

¹⁴⁰ Vyhovie v podstate akýkoľvek PAL, alebo GAL obvod s potrebným počtom I/O vývodov. Napr. GAL20V10-Q25 má oproti iným typom len 25% spotrebu energie a dopravné oneskorenie logickej hodnoty z vstupu na výstup max. 25ns.

Obrázok 166, Priebiehy napätí a prúdov 3-fázového usmerňovača pri $\alpha=0^\circ$



Obrázok 167, Časové priebehy napätí a prúdov v oblasti normálnych zaťažení



Rozbor úlohy: Hlavnou úlohou tohto regulátora je generovať zapínacie impulzy pre tyristory s fázovým posuvom teoreticky 0° až 180° elektrických. Keďže ale pri tyristoroch musí dochádzať ku vzájomnej komutácii, musíme znížiť riadiaci uhol o cca. 30° elektrických, aby bola zabezpečená spoľahlivá vzájomná komutácia tyristorov. Zapínací impulz pre príslušný prvok je odvodený od príslušnej fázy napájacieho napätia označenej vo obrázku ako SL1(V6), SL2(V8) a SL3(V7). Na výstupe usmerňovača je napätie označené červenou farbou a prúdy príslušných tyristorov sú označené farebne rovnako ako pri napätiach. Zapínací impulz musí byť súčasne privedený na tyristor nachádzajúci sa v kladnej skupine a zároveň na niektorý tyristor v zápornej skupine. Generovanie zapínacích impulzov môže vykonávať mikroprocesor, alebo programovateľný logický obvod PLD. Obvod PLD je výhodné použiť všade tam, kde je potrebné podmieňovať generovanie impulzov, prípadne meniť dĺžku a frekvenciu impulzov napr. zámena jediného impulzu za sériu impulzov.

Závislosť výstupného napätia usmerňovača na riadiacom uhle α :

$$U_d = \frac{3}{\pi} \int_{\frac{\pi}{3} + \alpha}^{\frac{2\pi}{3} + \alpha} \sqrt{6} \cdot U_f \cdot \sin(\nu) d\nu = \frac{3 \cdot \sqrt{6}}{\pi} \cdot U_f \cdot \cos(\alpha) \quad (24.)$$

$$U_d = \frac{3 \cdot \sqrt{6}}{\pi} \cdot U_f \cdot \cos(\alpha) \quad (25.)$$

kde U_f je efektívna hodnota fázového napätia

Logická závislosť generovania zapínacích impulzov:

```
I1P=((I1 & SL1)|(I3 & /SL3 & DIR)|(I2 & /SL2 & /DIR));  
I1N=((I1 & /SL1)|(I3 & SL3 & DIR)|(I2 & SL2 & /DIR));  
I3P=((I3 & SL3)|(I2 & /SL2 & DIR)|(I1 & /SL1 & /DIR));  
I3N=((I3 & /SL3)|(I2 & SL2 & DIR)|(I1 & SL1 & /DIR));  
I2P=((I2 & SL2)|(I1 & /SL1 & DIR)|(I3 & /SL3 & /DIR));  
I2N=((I2 & /SL2)|(I1 & SL1 & DIR)|(I3 & SL3 & /DIR));
```

I1P až I3P impulzy pre tyristory v kladnej skupine usmerňovača

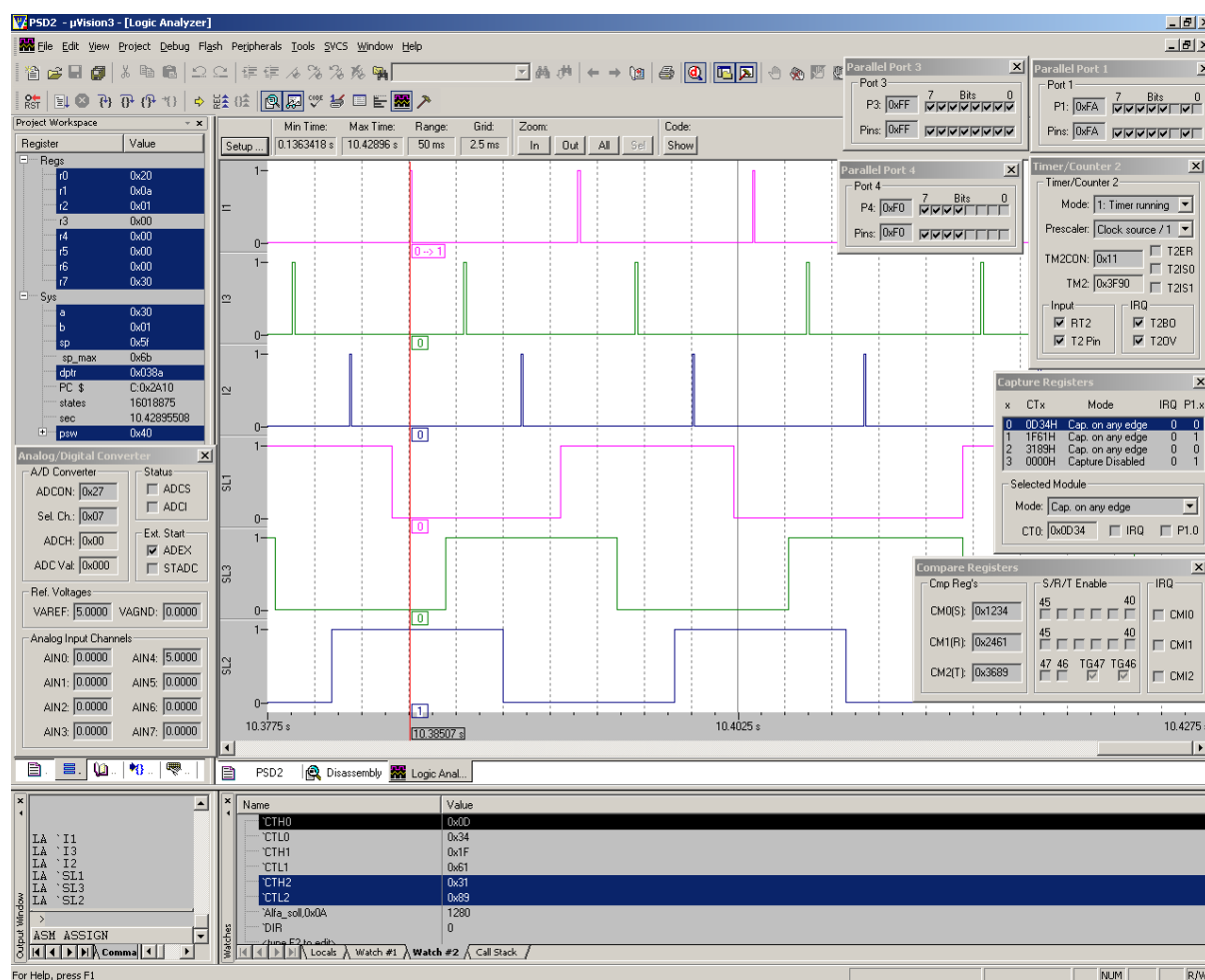
I1N až I3N impulzy pre tyristory v zápornej skupine usmerňovača

SL1 až SL3 tvarované signály jednotlivých fázových napätí napájacej siete

I1 až I3 primárne zapínacie impulzy

DIR signál smeru otáčania sa fázových napätí

Obrázok 168, Principiálne generovanie primárnych zapínacích impulzov pre tyristory

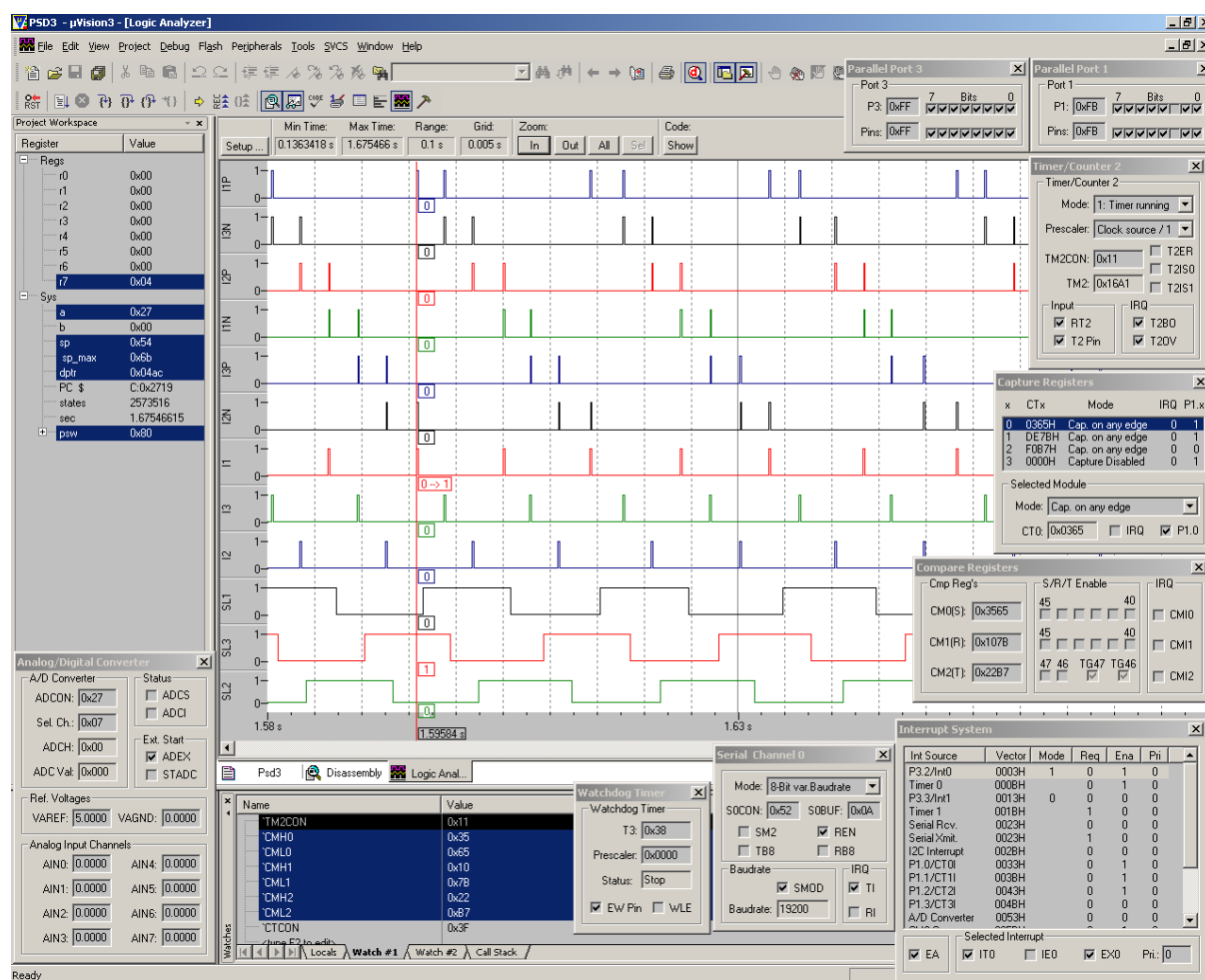


Poznámka autora: Celkovo boli zostrojené dva nezávislé regulátory s rovnakou funkciou. Jeden obsahoval analógové obvody (generátory píl, komparátory) ktoré spolupracovali s mikroprocesorovou jednotkou a druhá bola zostrojená len v diskretnom tvare.¹⁴¹

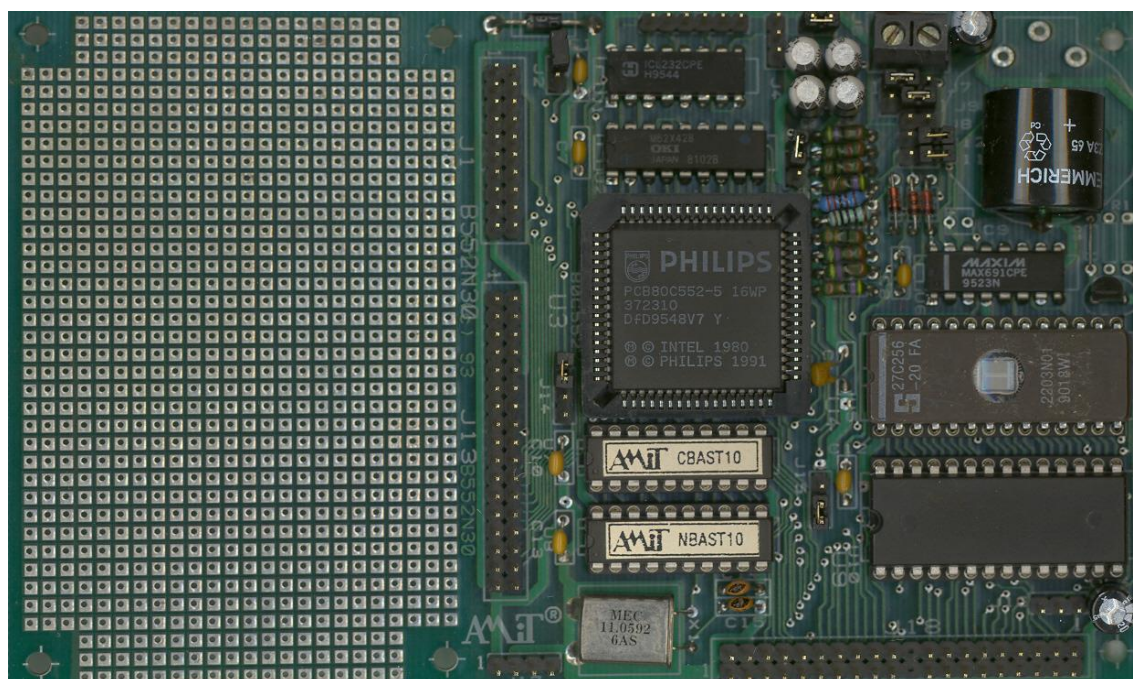
Všetky analógové obvody boli nahradené diskretnými obvodmi (generátory pílového napätia – časovačom T2, komparátory – číslicovými komparačnými jednotkami ...). Meranie periódy sieťového napätia sa v diskretnom regulátore vykonávajú záchytnou jednotkou CAPTURE tvorenou registrami CTHx a CTLx. Generovanie primárnych a sekundárnych impulzov s fázovým posuvom riadiaceho uhla α je vytvorené pomocou komparačnej jednotky CAPCOM tvorenou registrami CMHx a CMLx. Diskretný regulátor bol neskôr doplnený aj o automatickú korekciu riadiaceho uhla α pri zmene frekvencie t.j. periódy sieťového napätia v rozsahu frekvencií 40 až 70Hz.

¹⁴¹ Hlavnými dôvodmi prečo sa analógový regulátor nahradil diskretným je niekoľko, ale najväčším problémom bola vysoká teplota (približne 75°C) okolitého prostredia v blízkosti taviacich pecí, ktorá spôsobovala častú poruchovosť regulátora taviacich pecí.

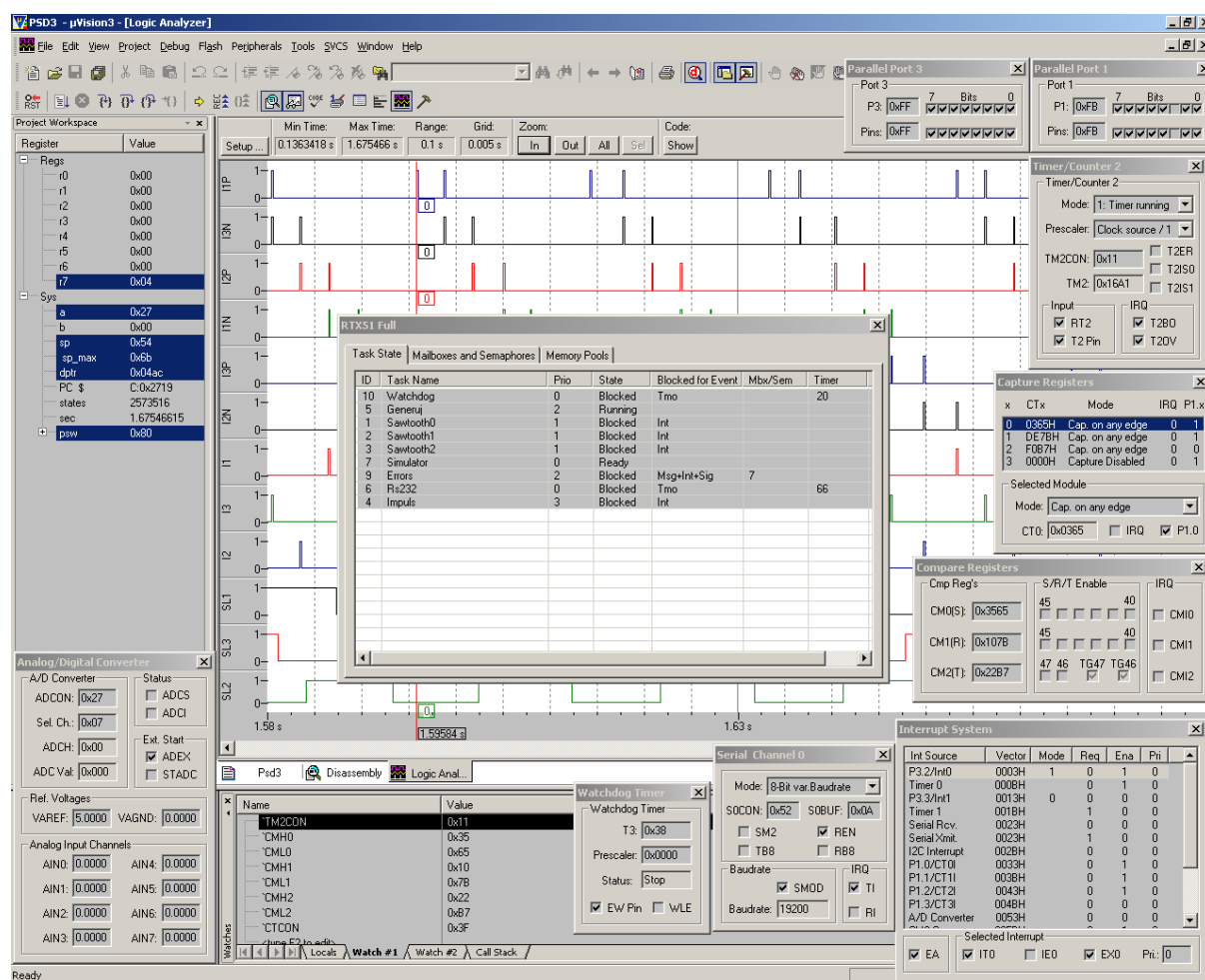
Obrázok 169, Generovanie impulzov v mikroprocesore 87C552



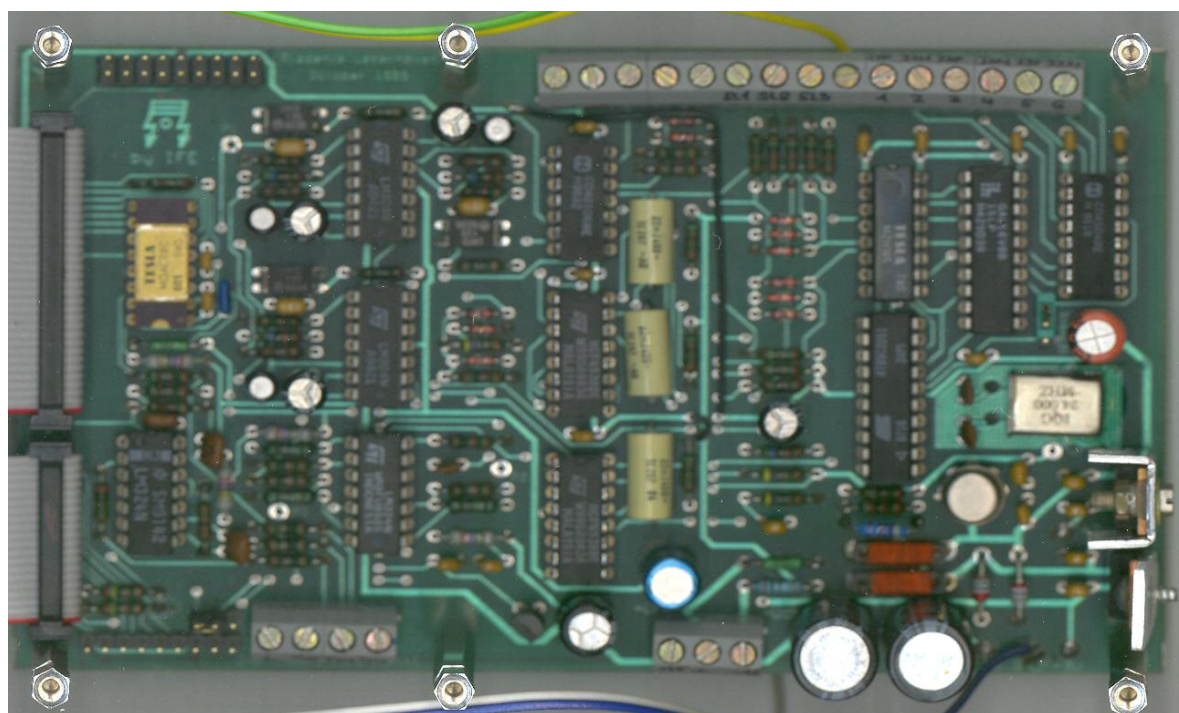
Obrázok 170, Vývojová doska diskretného riadenia usmerňovača s 80C552



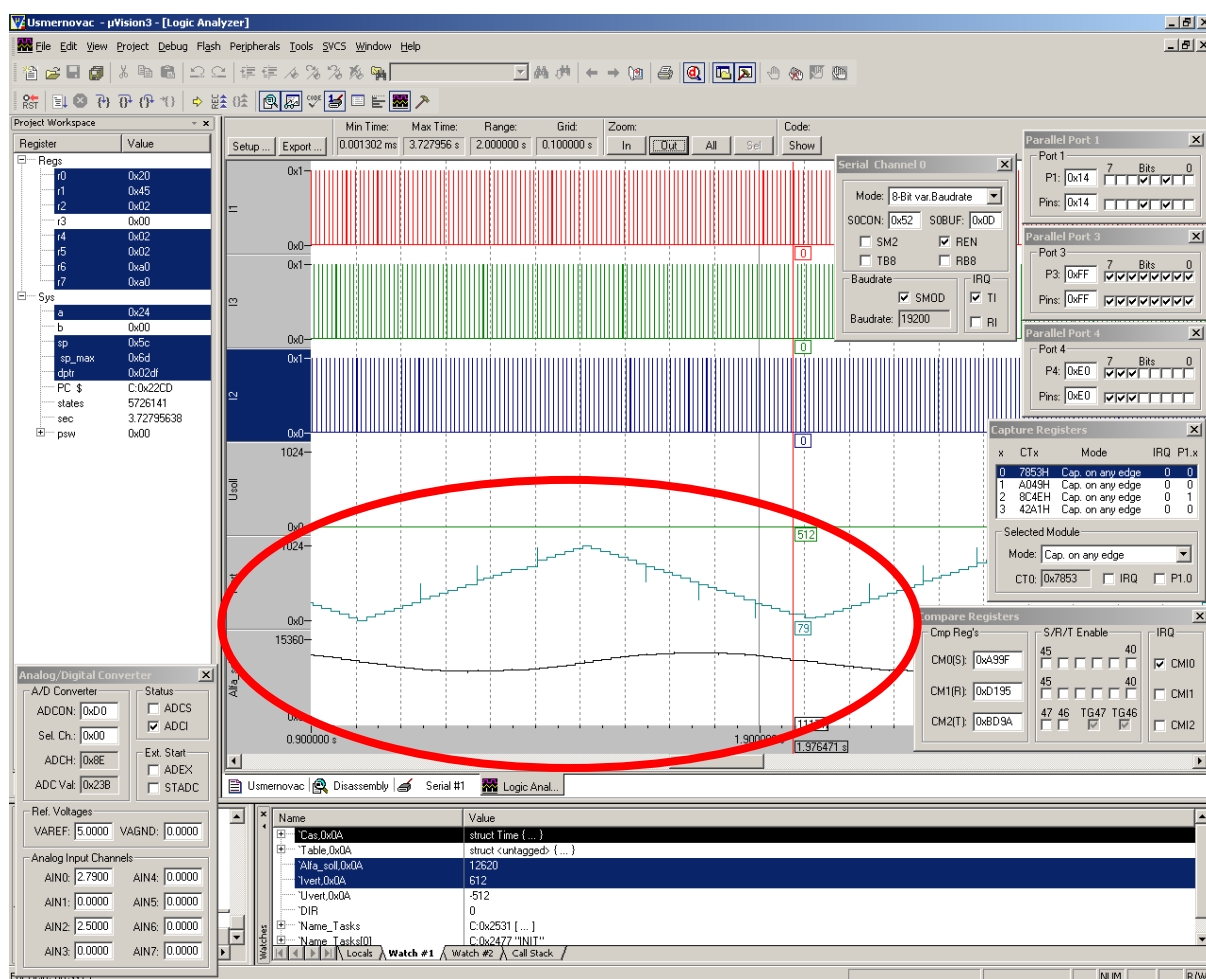
Obrázok 171, Zoznam úloh diskrétného regulátora usmerňovača pre tyristory



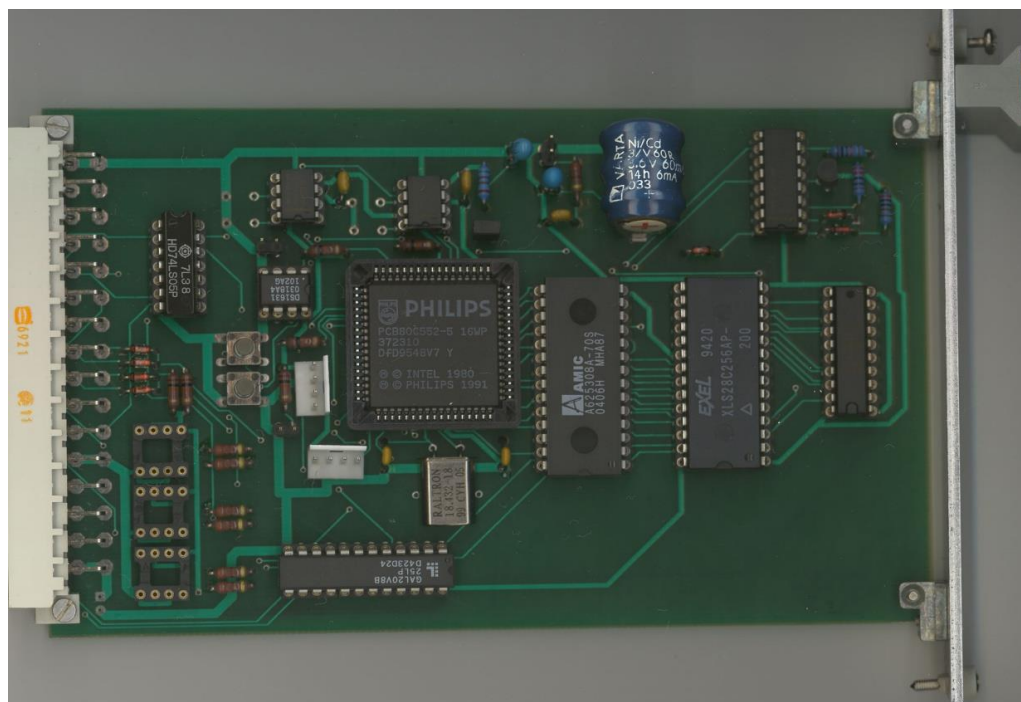
Obrázok 172, Interface riadenia usmerňovača s podpornými obvodmi variantu č.1



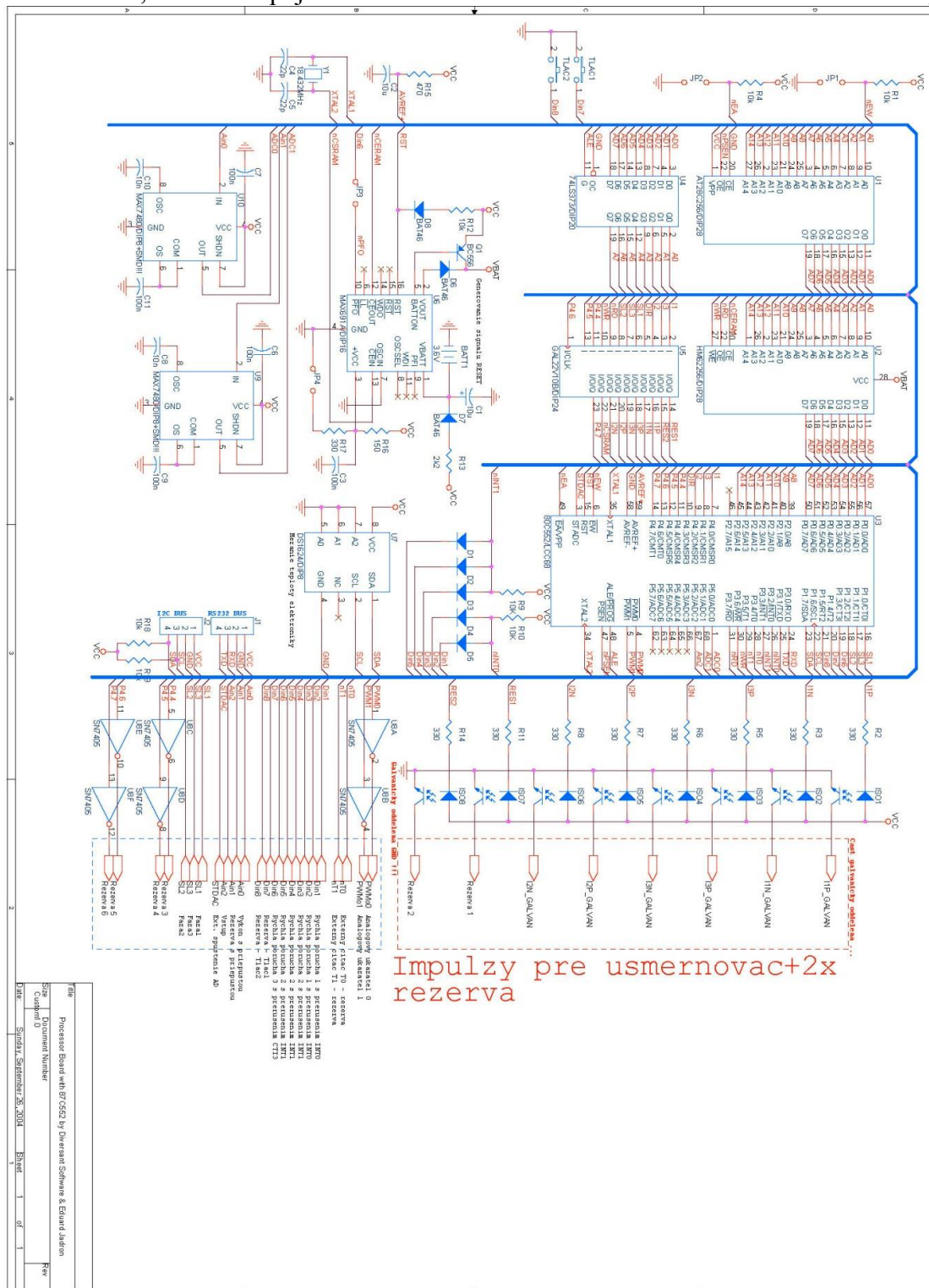
Obrázok 173, Simulácia regulátora diskretného riadenia s 80C552 varianty č.2



Obrázok 174, Doska diskretného riadenia usmerňovača s 80C552 varianty č.2



Obrázok 175, Schéma zapojenia diskretného riadenia¹⁴² usmerňovača 80C552 varianty č.2



¹⁴² V tejto variante bol použitý PSD prírastkový algoritmus s reálnou aritmetikou. Ďalšie varianty už obsahovali PSD regulátor s celočíselnou aritmetikou z dôvodu vylepšenia stability regulačnej slučky regulátora.

Zápis časti programu pre regulátor usmerňovača.

C:\Omega\data\Dropbox\Záloha\C51\termel\Usmernovac\Usmernovac.c

```
157 #define TIME 3 //Potrebne pre os_wait v Sawtooth0-2, idealna
    hodnota je 2 az 4 ...
158
159 void Sawtooth0 (void) _task_ SAWTOOTH0 _priority_ 1 //Priorita pre SAWTOOTHx, mala by byt
    vacsia ako 0 t.j 1 az 2, inac presne negeneruje cislicove pily !!!
160 {
161     int Temp;
162     TM2CON=0x01; //T2 s frekvenciou fosc/1 pretecie za 1*40.27ms
163     CTCON |= 0x02 + 0x01; //Max=(150/180)*Pocet 12800, Min=(30/180)*Pocet 2560
164     os_attach_interrupt (6);
165     while (1)
166     {
167         Watchdog;
168         switch (os_wait (K_INT+K_TMO,K*TIME,NULL)) //Cakam na fazu A a zachytavam obsahy zachytnych
            registrov ktore vyjadruju dlzku 1/2 periody fazoveho napatia siete
169         {
170             case INT_EVENT : CTI0=0; BUSY=1; Temp=Alfa_soll +(int)(256*CTH0+CTL0); CMH0=high (Temp); CML0
                =low (Temp); BUSY=0; break; //Prisla hrana fazy, tak vykonam toto ...
171             case TMO_EVENT : os_send_message (ERROR_MAIL,SAWTOOTH0,TIME); break;
            //Ak sa tu dostanem neprisla hrana fazy do TIME tak vyvolam chybu ...
172         }
173     }
174 }
175
176 void Sawtooth1 (void) _task_ SAWTOOTH1 _priority_ 1 //Tento task sa vyvola do cca. 400us z
    tasku s nizsou prioritou
177 {
178     int Temp;
179     CTCON |= 0x08 + 0x04;
180     os_attach_interrupt (7);
181     while (1)
182     {
183         Watchdog;
184         switch (os_wait (K_INT+K_TMO,K*TIME,NULL)) //Cakam na fazu C a zachytavam obsahy zachytnych
            registrov ktore vyjadruju dlzku 1/2 periody fazoveho napatia siete
185         {
186             case INT_EVENT : CTI1=0; BUSY=1; Temp=Alfa_soll +(int)(256*CTH1+CTL1); CMH1=high (Temp); CML1=
                low (Temp); BUSY=0; break; //Prisla hrana fazy, tak vykonam toto ...
187             case TMO_EVENT : os_send_message (ERROR_MAIL,SAWTOOTH1,TIME); break;
            //Ak sa tu dostanem neprisla hrana fazy do TIME tak vyvolam chybu ...
188         }
189     }
190 }
191
192 void Sawtooth2 (void) _task_ SAWTOOTH2 _priority_ 1
193 {
194     int Temp;
195     CTCON |= 0x20 + 0x10;
196     os_attach_interrupt (8);
197     while (1)
198     {
199         Watchdog;
200         switch (os_wait (K_INT+K_TMO,K*TIME,NULL)) //Cakam na fazu B a zachytavam obsahy zachytnych
            registrov ktore vyjadruju dlzku 1/2 periody fazoveho napatia siete
201         {
202             case INT_EVENT : CTI2=0; BUSY=1; Temp=Alfa_soll +(int)(256*CTH2+CTL2); CMH2=high (Temp); CML2
                =low (Temp); BUSY=0; break; //Prisla hrana fazy, tak vykonam toto ...
203             case TMO_EVENT : os_send_message (ERROR_MAIL,SAWTOOTH2,TIME); break;
            //Ak sa tu dostanem neprisla hrana fazy do TIME tak vyvolam chybu ...
204         }
205     }
206 }
207
208 sbit DIR = P4^3;
209
210 void OrderPhase (void) _task_ ORDERPHASE _priority_ 0
211 {
212     CTCON |= 0x20;
213     os_attach_interrupt (8); //Citlivost nastavim na log. "0"
214     while (1)
215     {
216         Watchdog;
217         switch (os_wait (K_INT+K_TMO,K*25,NULL))
218         {
```

C:\Omega\data\Dropbox\Záloha\C51\termel\Usmernovac\Usmemovac.c

```
219     case INT_EVENT : CTI2=0; if ((P1 & 0x07) != 1) DIR=RIGHT; else DIR=LEFT; os_send_signal (
    INIT); break;
220     case TMO_EVENT : os_send_message (ERROR_MAIL, ORDERPHASE, TIME); break;
                        //Neprišli mi fazy do 500ms, tak vyhlásim ERROR ...
221 }
222 CTRCON &= 0x20;
223 os_detach_interrupt (8);
224 os_delete_task (ORDERPHASE);
225 }
226 }
227
228 void Delay (unsigned char Dlzka) //reentrant
229 {
230     while (-- Dlzka != 0x00);
231 }
232
233 #pragma REGISTERBANK (3)
234
235 sbit I1 = P4^0;
236 sbit I3 = P4^1;
237 sbit I2 = P4^2;
238
239 void Impuls (void) _task_ IMPULS _priority_ 3
240 {
241     #define Length 75 //60*1.366=85us pre dlzku
    impulsu !!! ak nemam funkciu Delay reentrantnu
242     #define Dlzka (Length/1.366) //Aby som nemusel stale
    prepocitavat na skutocnu dlzku impulsu ...
243     os_attach_interrupt (11);
244     os_attach_interrupt (12);
245     os_attach_interrupt (13);
246     while (1)
247     {
248         Watchdog;
249         switch (os_wait (K_TMO+K_INT, K*100, NULL))
250         {
251             case INT_EVENT : switch (TM2IR & 0x70) //RTX51 HANDLER prepne do cca.
    110us na tento task zovsadiat !!!
252             {
253                 case 0x10 : CMI0=0; I1=1; Delay (Dlzka); I1=0; break; //Generujem 100us
    impuls I1 pre SL1 pre podmienku CMH0+CML0=TMH2+TML2
254                 case 0x20 : CMI1=0; I3=1; Delay (Dlzka); I3=0; break; //Generujem 100us
    impuls I3 pre SL3 pre podmienku CMH1+CML1=TMH2+TML2
255                 case 0x40 : CMI2=0; I2=1; Delay (Dlzka); I2=0; break; //Generujem 100us
    impuls I2 pre SL2 pre podmienku CMH2+CML2=TMH2+TML2
256                 default : TM2IR &= 0x70; break; //Ak som tu tak z
    nevysvetliteľných dôvodov prišli niektoré, alebo všetky dve-tri prerušenia od komparátora !!!
    Stane sa to pri poruchách na silových fázach a vtedy nesmie generovať impuls !!! ...
257             }
258             break;
259         case TMO_EVENT : os_send_message (ERROR_MAIL, IMPULS, TIME); break;
260         }
261     }
262 }
```

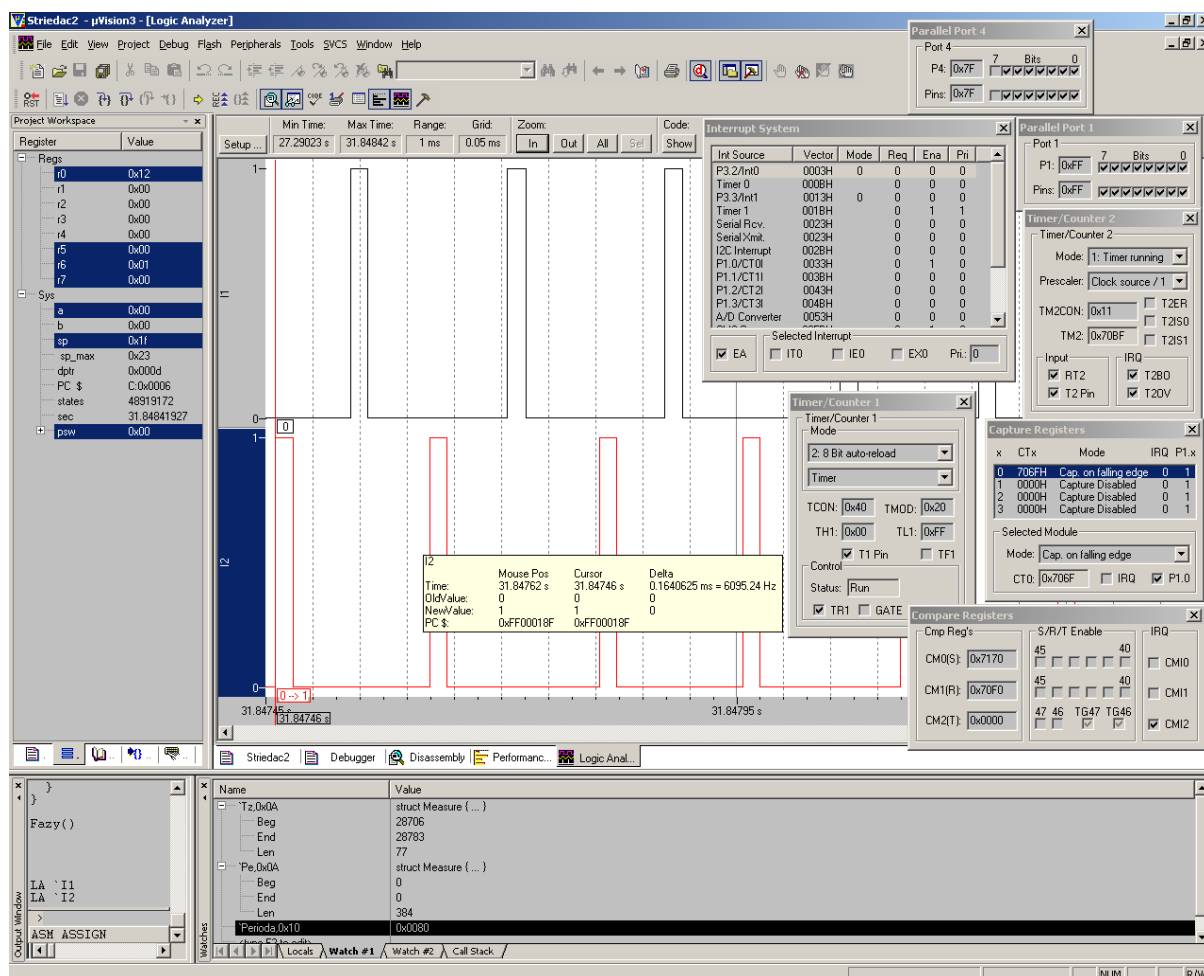
17.7. Diskrétné riadenie pre tyristorové striedače s rezonančným obvodom

Hlavnou úlohou tohto regulačného obvodu bolo riadenie chodu striedača s rezonančným obvodom a s nepretržitou kontrolou minimálnej doby zatvorenia t_z , ktorá musí byť pre správnu funkciu minimálne $t_z \geq 3 \cdot t_q$ ($\mu\text{s}; \mu\text{s}$). Čas t_q je katalógová hodnota, ktorá udáva vypínací čas tyristora. Vzájomný pomer periódy výstupného napätia T a doby zatvorenia tyristora t_z sa musia nachádzať pre optimálnu prevádzku striedača približne v rozsahu empiricky zisteného a udávaného δ .

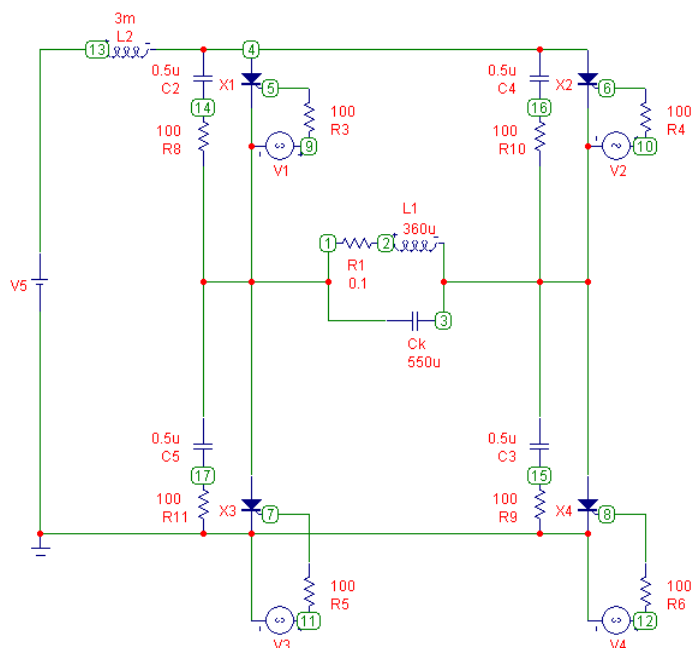
$$\delta \approx \frac{T}{t_z} \approx 6.3 \quad (1; s, s) \quad (26.)$$

Dĺžka generovaného zapínacieho impulzu pre tyristor striedača bude minimálne $t_{onmin} = 18\mu\text{s}$ s možnosťou nastavenia až na $t_{onmax} = 100\mu\text{s}$. Diskrétny regulátor striedača musí byť schopný generovať zapínacie impulzy až do frekvencie $f_{max} = 6000\text{Hz}$.

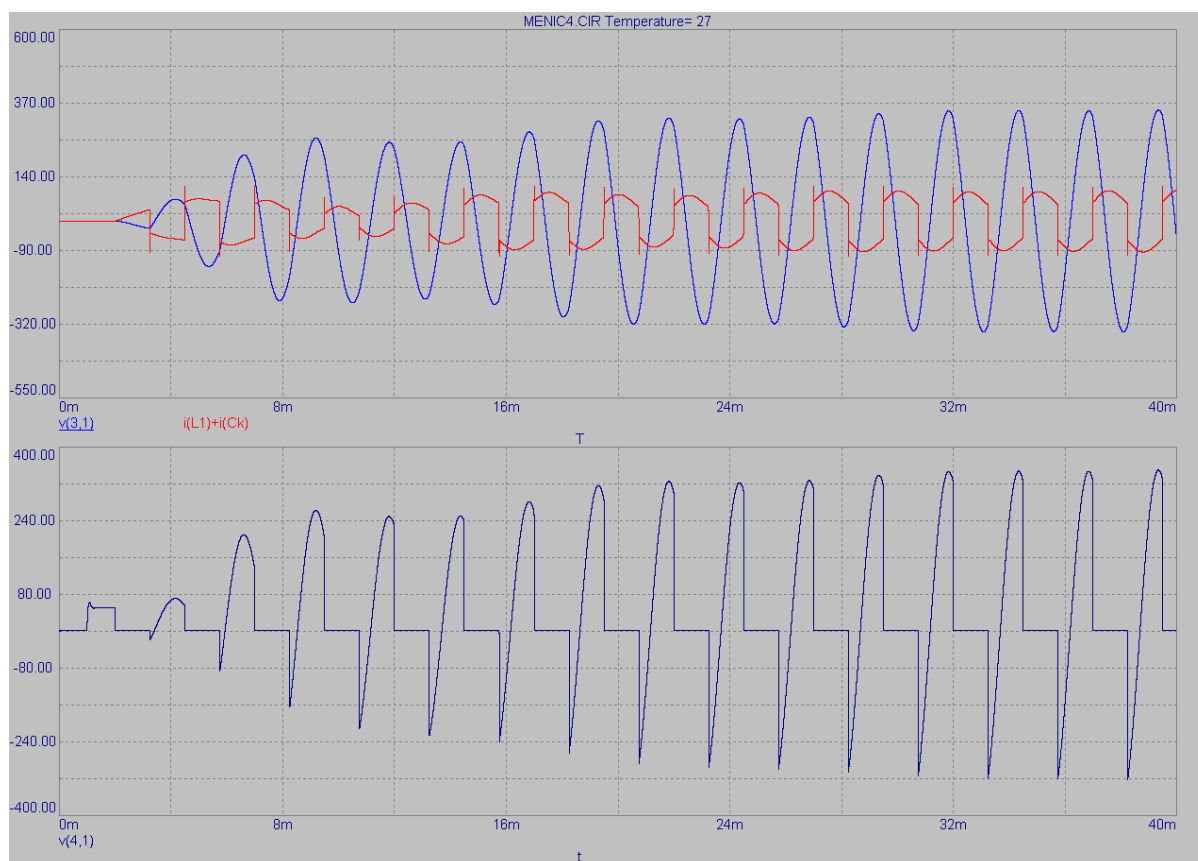
Obrázok 176, Generovanie zapínacích impulzov pre striedač v mikroprocesore 87C552



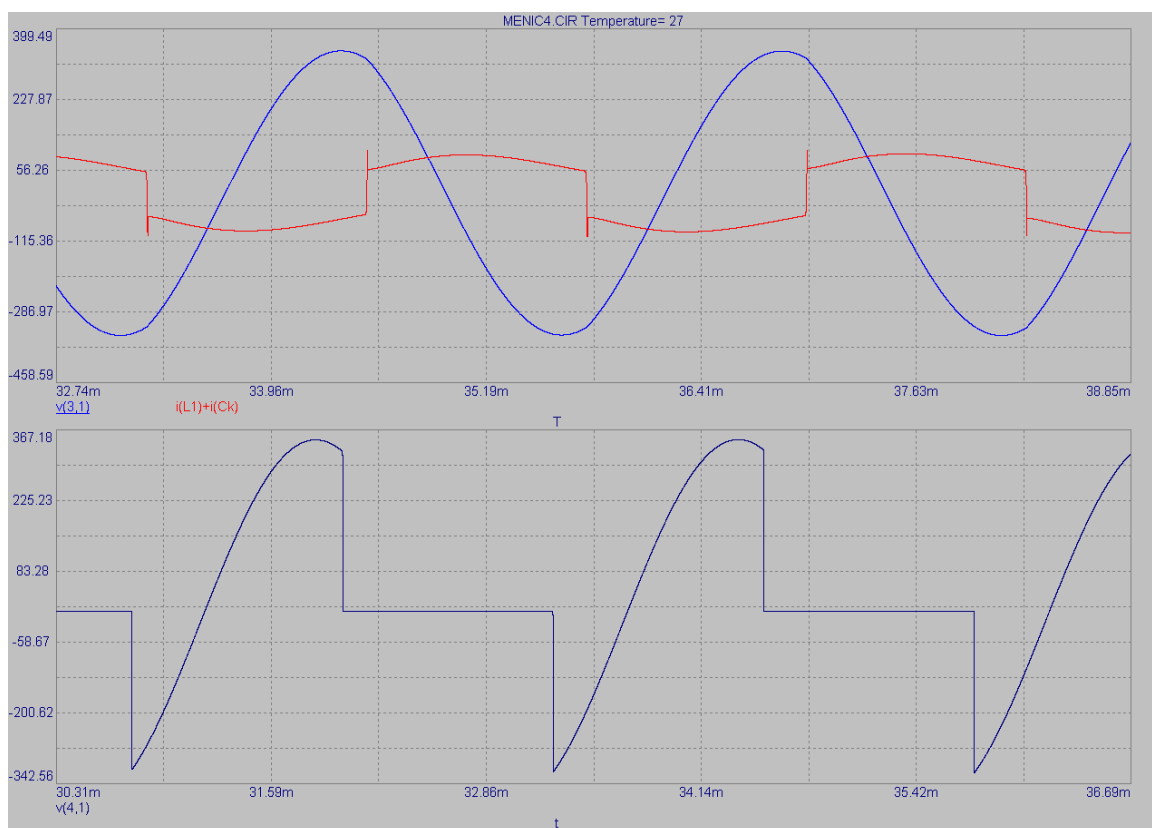
Obrázok 177, Zjednodušený model tyristorového striedača s rezonančným obvodom



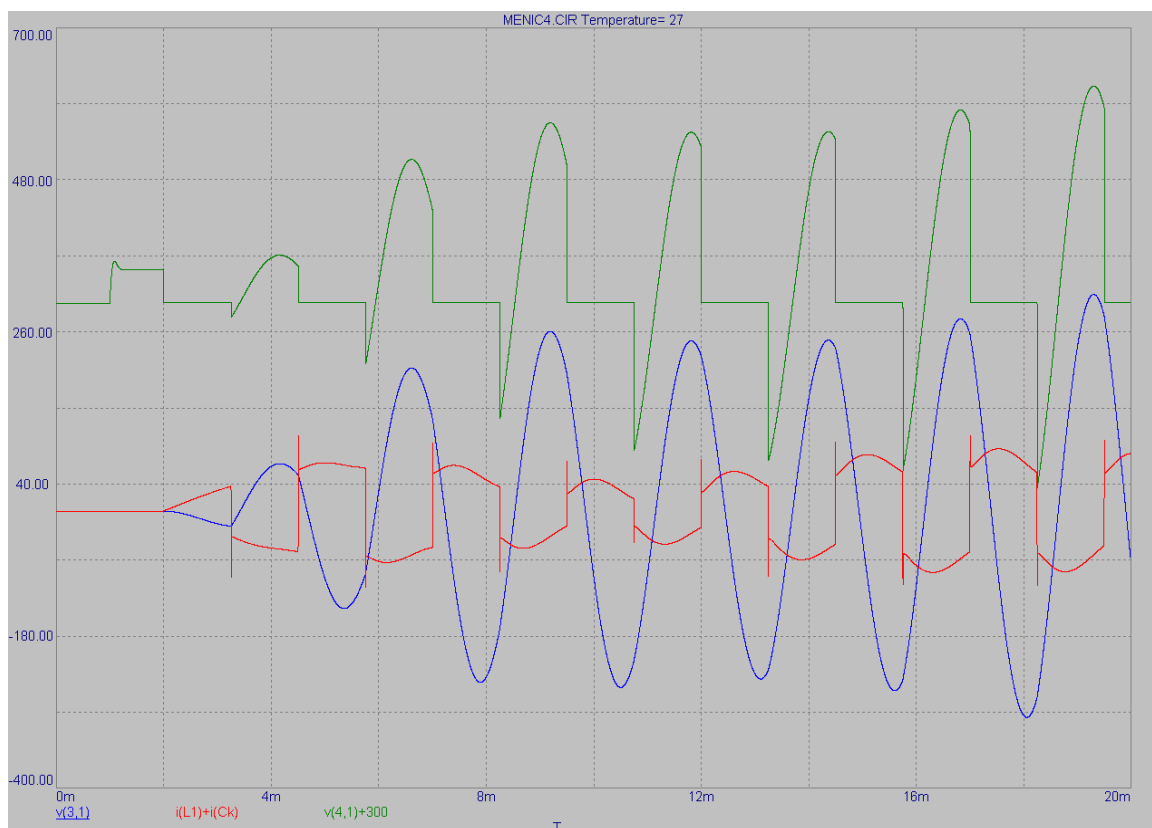
Obrázok 178, Grafické priebehy napätí v tyristorovom striedači s rezonančným obvodom



Obrázok 179, Grafické priebehy napätí v tyristorovom striedači s rezonančným obvodom



Obrázok 180, Detailné priebehy napätí v tyristorovom striedači s rezonančným obvodom



17.8. Spracovanie výsledkov meraní na striedači.

Po zostrojení striedača s diskretným regulátorom boli namerané tieto veličiny na tyristoroch TR961-500-12-NML s časom komutácie $t_q = 170\mu s$, pracovná frekvencia $f_{rez} = 413Hz$. Regulátor bol navrhnutý tak, aby komutačný uhol tyristora striedača bol maximálne $\beta_{el} = 26^\circ$ elektrických, čo je v súlade s nižšie uvedeným vzťahom.

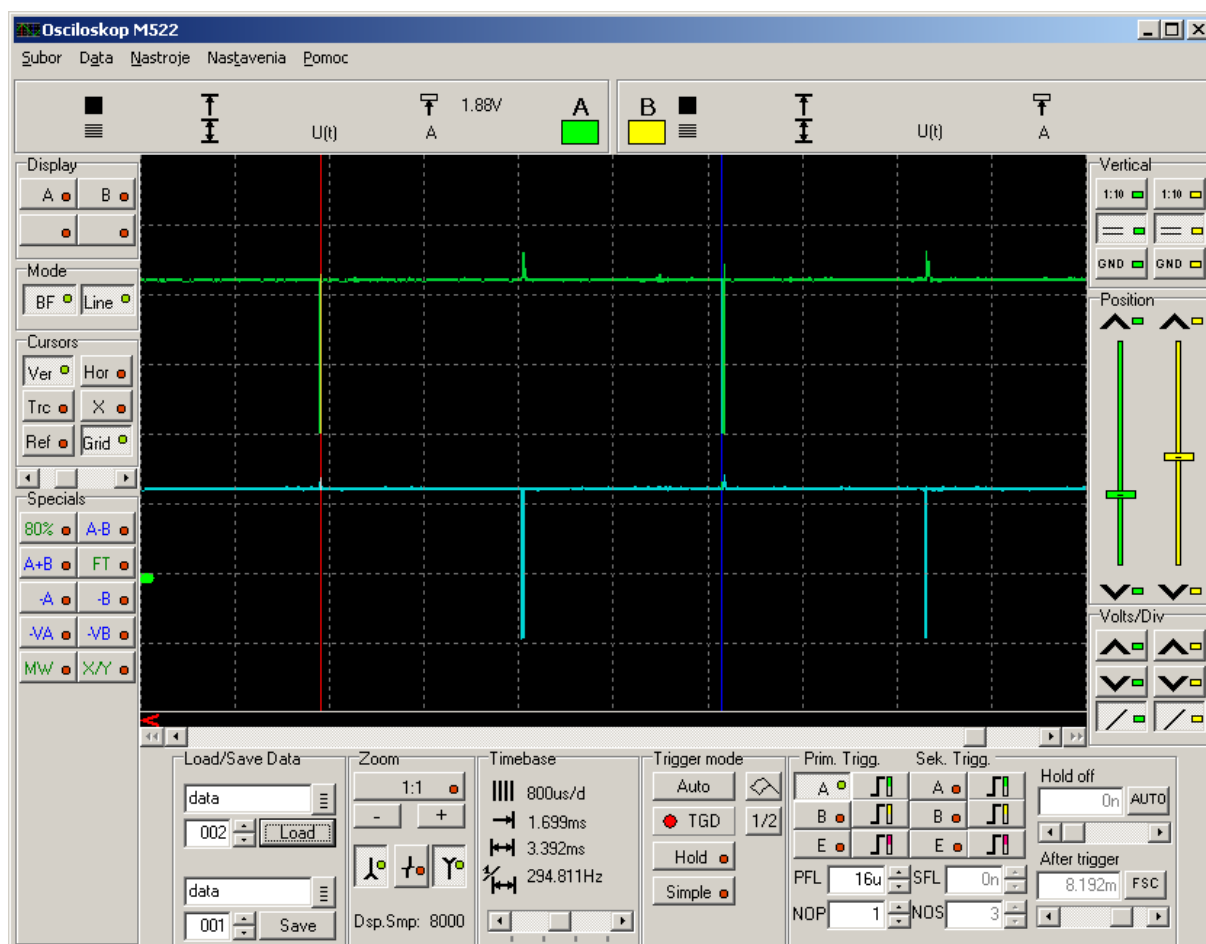
Po dosadení vyššie uvedených veličín do vzťahu vidíme, že regulátor pracuje správne

$$\beta_{el} = 2 \cdot \pi \cdot f_{rez} \cdot t_q \quad (rad \cdot s^{-1}; Hz, s) \quad (27.)$$

Po prevedení na stupne vynásobením $\beta_{el} \cdot \frac{180}{\pi}$

$$\beta_{el} = 360 \cdot f_{rez} \cdot t_q = 360 \cdot 413 \cdot 170 \cdot 10^{-6} = 25,27^\circ \quad (^\circ; Hz, s) \quad (28.)$$

Obrázok 181, Zapínacie impulzy pre jednotlivé diagonály tyristorov



Program v jazyku C51 pre tyristorový striedač

C:\Omega\data\Dropbox\Záloha\C51\termel\Striedac2\Striedac2.c

```
1  #include <reg552.h>
2  #include <intrins.h>
3  #include <absacc.h>
4  #include <stdlib.h>
5  #include <stdio.h>
6  #include <rtx51tny.h>
7  #include <dsb.h>
8
9  #define SIMULx
10
11 #define REGULATOR 0
12 #define ERRORS 1
13
14 //-----
15 #define XTAL 18.432E6
16 #define Frekvencia_horna 700 //478
17 #define Frekvencia_dolna 300
18 #define Cislo (XTAL/(12*2*Frekvencia_horna))
19 #define Tzmin ((XTAL*100E-6)/12) //Tzmin=100us
20 #define Hranica_Dolna (XTAL/(12*2*(Frekvencia_dolna))) // Dolna frekvencia (zaraska dolnej
    frekvencie)
21 #define Hranica_Horna (XTAL/(12*2*(Frekvencia_horna)))
22 //-----
23
24 #define WLE 0x10
25 #define INTERVAL 200 //Watchdog cca. 200ms
26 #define Watchdog ((PCON|WLE),(T3=0x100-INTERVAL))
27 #define DCapture (IEN1&=~0x01) //Zakazem akceptovanie pereusenja od Capture
    jednotky, kde je Tz
28 #define ECapture (IEN1|=0x01) //Povolim akceptovanie pereusenja od Capture
    jednotky, kde je Tz
29 #define DCompare //(IEN1&=~0x10) //Pozastavim pocas vypoctu Perioda++ a Perioda--
    generovanie impulzu v Compare jednotke
30 #define ECompare //(IEN1|=0x10) //Povolim Compare jednotku
    //-----
31
32 sbit I1 = P4^0;
33
34 int Per = 0x0000, Perioda = Cislo;
35
36 void Impuls1(void) interrupt 11 using 2
37 {
38     #define Dlzka 4 //Dlzka = 4 cca.20us s ostatnym kodom. Tu sa meni scasti dlzka impulzu !!!
39     unsigned char Temp=Dlzka;
40     I1=1;
41     Per+=Perioda;
42     CMH0=high(Per);
43     CML0=low(Per); //Generujem pre podmienku CMH0+CML0=TMH2+TML2
44     Watchdog;
45     while(--Temp!=0x00);
46     I1=0;
47     CMI0=0;
48 }
49
50 struct Measure {int Beg,End,Len;}; //Definujem strukturu pre meranie Tz a Pe
51 struct Measure Tz = {0x0000,0x0000,Tzmin}; //Uchovavam Tz
52 struct Measure Pe = {0x0000,0x0000,(int)(Tzmin*6.3)}; //Uchovavam dlzku peridy Pe
53
54 void Capture(void) interrupt 6 using 3
55 {
56     #define CTP0 0x01
57     #define CTN0 0x02
58     switch (CTCON & (CTP0+CTN0))
59     {
60         case CTP0 : Tz.End=Pe.Beg=(256*CTH0+CTL0); Tz.Len=(Tz.End-Tz.Beg); break;
61         case CTN0 : Tz.Beg=Pe.End=(256*CTH0+CTL0); Pe.Len=(Pe.End-Pe.Beg); break;
62     }
63     CTCON^=(CTP0+CTN0);
64     CTI0=0;
65 }
66
67 void Timer(void) interrupt 3 //Vsetko napisem v assembleri ???
```

C:\Omega\data\Dropbox\Záloha\C51\termel\Striedac2\Striedac2.c

```
68 {
69     P1 ^= 0x01;
70 }
71
72 void Initialize (void)
73 {
74     CMH0 = high (Per);
75     CML0 = low (Per);
76     IEN1 |= 0x10; //Povolim komparacnu jednotku
77     IEN1 |= 0x01; //Povolujem Capture jednotku
78     IP1 |= 0x10; //Komparacna jednotka s vacsou prioritou ...
79     TM2CON = 0x01; //T2 s frekvenciou fosc/1 pretecie za 1*40.27ms
80     CTCON |= CTNO; //Citlivost na prechod "0" CAPTURE 1
81 }
82
83 sbit ZAVES_SA = P3^5; //Od tohoto okamihu znižujem
84 sbit SOM_ZAVESENY = P3^4;
85
86 //-----
87 #define Kp 2.800 //Proporcionálna konštanta Kp
88 #define dT 0.005 //Interval vzorkovania pe regulator dT=20ms
89 #define Ti 0.041 //Integracna konštanta Ti, je potrebné ju
    prehodnotiť pri zmene menica ...
90 #define Td 0.001 //Derivacna konštanta Td, ovplyvňuje strmý
    nárast pri zmene reg. odchylky, všim si totiž strmý nárast odchylky !!!
91 //-----
92 float E0 = 0.000; //Regulacna odchylka E0 nova
93 float E1 = 0.000; //Regulacna odchylka E1 stara
94 float E2 = 0.000; //Regulacna odchylka E2 moc stara
95 float Y0 = 0.000; //Akčná veľkosť Y0 nova pre regulator
96 float Y1 = (float)Hranica_Horna; //Akčná veľkosť Y1 stara pre regulator
97 float KA = (Kp*(1.0+dT/(2.0*Ti))+(Td/dT)); //Konštanta KA prírastkového
    regulatora
98 float KB = (Kp*(1.0-dT/(2.0*Ti))+(2.0*Td/dT)); //Konštanta KB prírastkového
    regulatora
99 float KC = (Kp*(Td/dT)); //Konštanta KC prírastkového regulatora
100 //-----
101
102 #define Pomer 4.500 //4.9 bolo OK ???
103
104 void Regulator (void) _task_ REGULATOR
105 {
106     unsigned char Counter0, Counter1;
107     float Pomer_vert; //Nastavenie Pomer_vert pri simulácii signálov
    programu START
108     bit R=0;
109     Initialize ();
110     #ifdef SIMUL
111     ET1=1;
112     TR1=1;
113     #endif
114     while (1)
115     {
116         os_wait (K_TMO, 5, NULL); //Čakám pokiaľ nepríde príkaz na uplynutie času
117         if (ZAVES_SA == 1)
118         {
119             Watchdog;
120             DCapture; //Zakážem na chvíľu prerušenie od CAPTURE
121             Pomer_vert = (float)Pe.Len/(float)Tz.Len; //Výpočet trva cca. 800us pri 18.432MHz
122             ECapture;
123             E0 = Pomer - Pomer_vert; //Výpočet trva cca. 131us pri
    18.432MHz
124             Y0 = Y1 + (KA * E0) - (KB * E1) + (KC * E2); //Výpočet trva cca.
    1064us pri 18.432MHz
125             if (Y0 >= Hranica_Dolna) Y0 = Hranica_Dolna;
126             if (Y0 <= Hranica_Horna) Y0 = Hranica_Horna;
127             DCompare;
128             Perioda = (int)Y0;
129             ECompare;
130             Y1 = Y0;
131             E2 = E1;

```

Page2

C:\Omega\data\Dropbox\Záloha\C51\termel\Striedac2\Striedac2.c

```
132     E1=E0 ;
133     if (Pomer_vert >Pomer )           R=1;
134     if (Pomer_vert <(Pomer -0.9)) R=0;
135     if (R=1) {Counter0 ++; Counter1 =0x00 ;}
136     if (R=0) {Counter1 ++; Counter0 =0x00 ;}
137     if (Counter0 >=5) {SOM_ZAVESENY =1; Counter0 =0x00 ; os_wait (K_TMO ,20 ,NULL);}
138     if (Counter1 >=20){ SOM_ZAVESENY =0; Counter1 =0x00 ; os_wait (K_TMO ,20 ,NULL);}
139     }
140     else
141     {
142         Watchdog ;
143         os_wait (K_TMO ,200 ,NULL);
144         Y0=Y1=( float )Cislo ;
145         Perioda =Cislo ;
146         SOM_ZAVESENY =0;
147         Counter0 =0;
148         Counter1 =0;
149     }
150 }
151 }
```

17.9. Skalárne riadenie asynchrónneho motora pomocou modulácie SWPWM

Príklad: Pomocou programovacieho jazyka A51, C51, alebo RTX51 vytvorte program, ktorý bude riadiť frekvenčný menič ktorý napája jednosmerný motor s cudzím budením, jednofázový komutátorový motor a trojfázový asynchrónny motor ďalej len ASM s klieťkovou kotvou. Daný menič musí byť schopný napájať jednosmerným a striedavým napätím s premenlivou veľkosťou napätia a frekvencie daný pohon. Napájacie napätie pre motor bude v rozsahu 0V po 400V_{AC} s frekvenciou 0 až 150Hz. Výstupy portu mikroprocesora C504 budú s pomocou 20kHz PWM ovládať trojfázové mostové zapojenie s IGBT, ktorý plní funkciu napäťového striedača. Striedač bude ovládaný pomocou dvojice tlačidiel. Dve tlačidlá budú slúžiť na voľbu otáčok motora a zároveň na zmenu smeru otáčania. Zároveň program doplníte o ovládanie pomocou sériového rozhrania RS232 s nasledujúcimi parametrami: 2400b.s⁻¹, 8 dátových bitov, 1 štart a stop bit, bez parity. Výstupné napätie bude modulované pomocou SWPWM (Sinusoidal Weight Pulse Width Modulation).

Obrázok 189 bližšie zobrazuje vytvorenie rotujúceho magnetického poľa statora asynchrónneho motora (ASM). Obrázok 190 až Obrázok 194 podrobnejšie znázorňujú problematiku riadenia ASM za pomoci mikroprocesora C504 s riadiacimi a regulačnými obvodmi.

Rozbor úlohy:

Pre otáčky asynchrónneho motora bude platiť vzťah:

$$\omega_1 = \frac{2 \cdot \pi \cdot f}{p} \quad (\text{rad} \cdot \text{s}^{-1}; \text{Hz}, 1) \quad (29.)$$

alebo

$$n_1 = \frac{60 \cdot f}{p} \quad (\text{ot} \cdot \text{min}^{-1}; \text{Hz}, 1) \quad (30.)$$

Pre magnetický tok v motore musí platiť vzťah:

$$\Phi = \frac{U_1}{4,44 \cdot f_1 \cdot N_v \cdot k_v} = k \frac{U_1}{f_1} \quad (\text{Wb}; \text{V}, \text{Hz}, 1, 1) \quad (31.)$$

Ak chceme zachovať konštantný magnetický tok stroja musí platiť:

$$\frac{U_1}{f_1} = \text{const} \quad (\text{V}, \text{Hz}, 1) \quad (32.)$$

Pre maximálny moment stroja bude teda platiť vzťah:

$$M_M = \frac{3 \cdot U_1^2}{2 \cdot \varpi_1 \cdot X_K} = K_1 \frac{U_1^2}{f_1^2} \quad (N \cdot m; V, rad \cdot s^{-1}, \Omega) \quad (33.)$$

Riadiaci algoritmus mikroprocesora musí ovládať striedač tak, aby bolo možné na jeho výstupných svorkách vygenerovať striedavé napätie ktorým je pohon napájaný. Pomocou PWM modulácie je nutné vyrobiť napätie sínusového priebehu. Z externej pamäti RAM mikroprocesora musí byť výber vzorky napätia každých $\Delta T = 50\mu s$ čo je hodnota napäťovej vzorky PWM modulátora tak, aby bolo možné vyrobiť napätia s požadovanou amplitúdou a frekvenciou. Použitie operačného systému RTX51 si musíme veľmi dobre uvážiť, pretože musíme zabezpečiť periodický výber hodnôt z pamäti RAM každých $50\mu s$ na každú fázu, čo je časovo veľmi náročné.

Východiskom z tohto problému by bolo použitie internej pamäti programu mikroprocesora, kde by sa nachádzali už vopred vypočítané vzorky napätia pre jednotlivé frekvencie asynchrónneho motora. Takto by sme zabezpečili uloženie vzoriek v rozsahu pracovných frekvencií 0 až 50Hz. Množstvo potrebnej pamäti programu potrebnej pre uloženie vzoriek by sme mohli vypočítať pomocou nižšie uvedeného vzťahu s tým, že budeme do pamäti ukladať len $\frac{1}{2}$ sínusového priebehu, pretože platí vzájomná symetria kladnej a zápornej polvlny.

Vzťah na výpočet množstva pamäti pre uloženie vzoriek PWM:

$$m = \sum_{i=1}^n \frac{1}{2.048 \cdot i} \cdot f_{PWM} \quad (kB; 1, Hz, Hz) \quad (34.)$$

n počet pracovných frekvencií ASM motora (-)

i i -tá pracovná frekvencia ASM motora (Hz)

f_{PWM} pracovná frekvencia PWM modulátora (Hz)

m množstvo pamäti programu pre vzorky (kB)

Pre $n = 50$ a $f_{PWM} = 20kHz$ bude potrebných 43.94kB pamäti programu.

Ak by sme ale chceli zvýšiť počet pracovných frekvencií ($n = 100$ a pri zachovaní f_{PWM}) ASM pohonu, vďaka symetrii sínusového priebehu by stačilo do pamäti ukladať len $\frac{1}{4}$ periódy napätia pre ASM motor. Takto by postačovalo využiť pre vzorky len

$m = \sum_{i=1}^n \frac{1}{2 \cdot 2.048 \cdot i} \cdot f_{PWM} = 25.33kB$ pamäti programu. Zvyšná pamäť by mohla byť využitá pre riadiaci program pohonu.

Najväčším problémom je množstvo vzoriek, ktoré sú ukladané do pamäti programu mikroprocesora C504. Ak chceme zachovať konštantný magnetický tok je výhodnejšie a rýchlejšie nahradiť operáciu násobenia operáciou sčítania podľa nasledujúceho vzťahu. Tento príklad je zjednodušený variant predchádzajúceho riešenia.

$$U = A \cdot \sin B = \cos(\arccos A) \cdot \sin B \quad (35.)$$

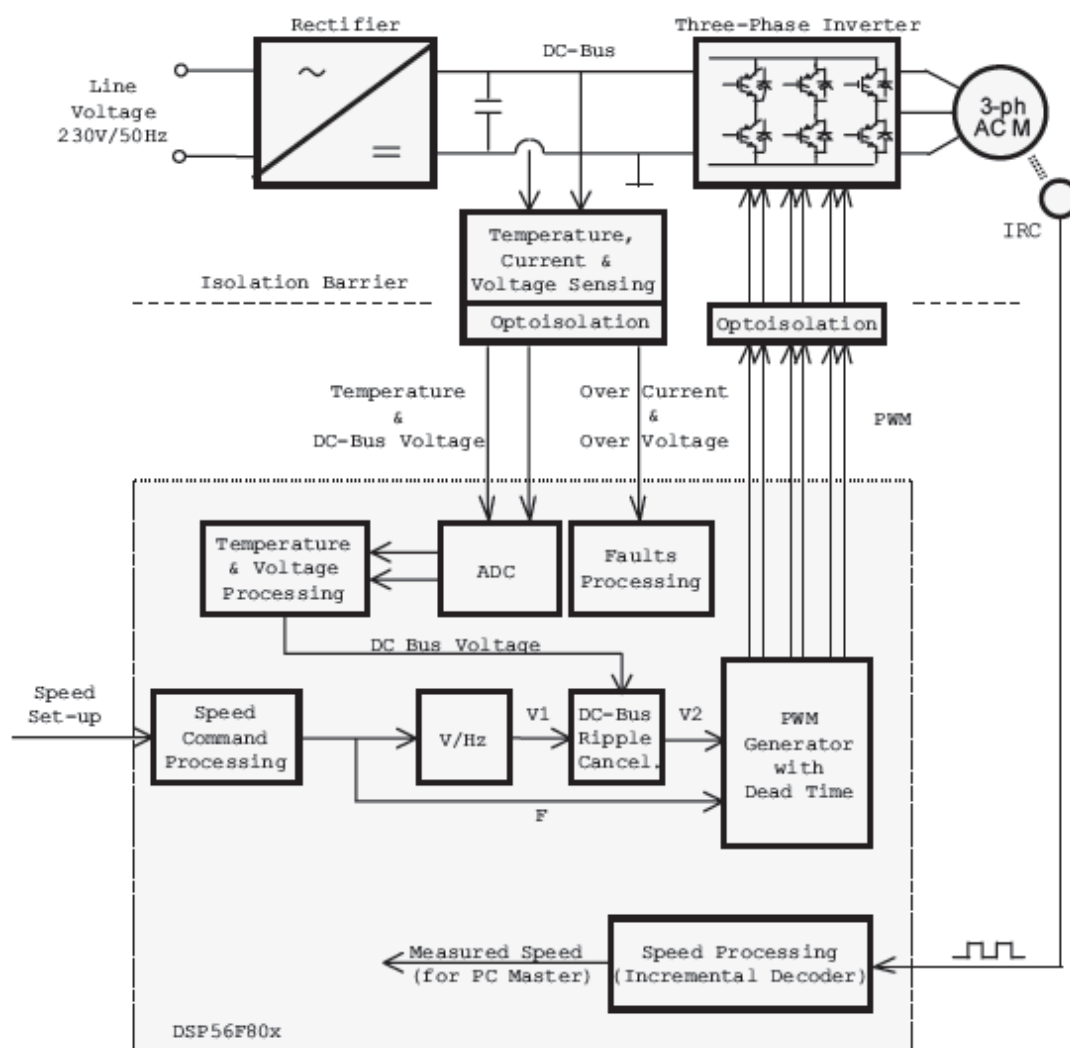
Ak bude $A' = \arccos A$, potom sa nahradí násobenie tzv. sčítacím teorémom

Po rozklade dostaneme vzťah:

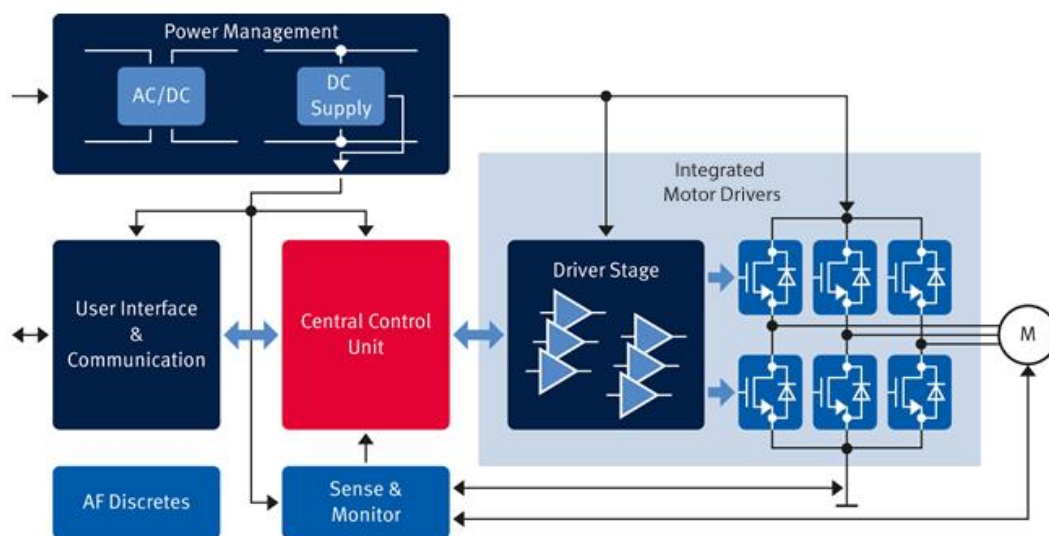
$$U = \cos A' \cdot \sin B = \frac{1}{2} \cdot [\sin(B - A') + \sin(B + A')] \quad (36.)$$

Kde B predstavuje vzorku napätia ktorá je uložená v tabuľke a je upravená o hodnotu A' , ktorá zabezpečí zmenu napätia na pohone tak, aby platil konštantný pomer $\frac{U}{f}$.

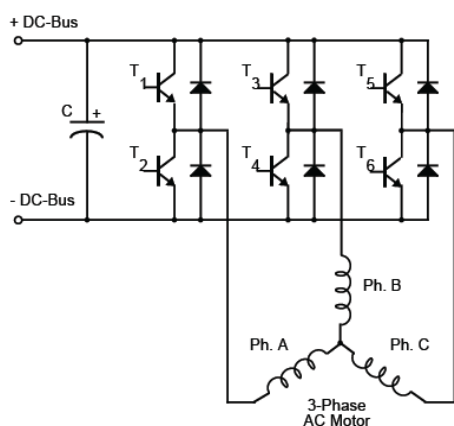
Obrázok 182, Schéma zapojenia skalárneho riadenia 3fázového ASM motora



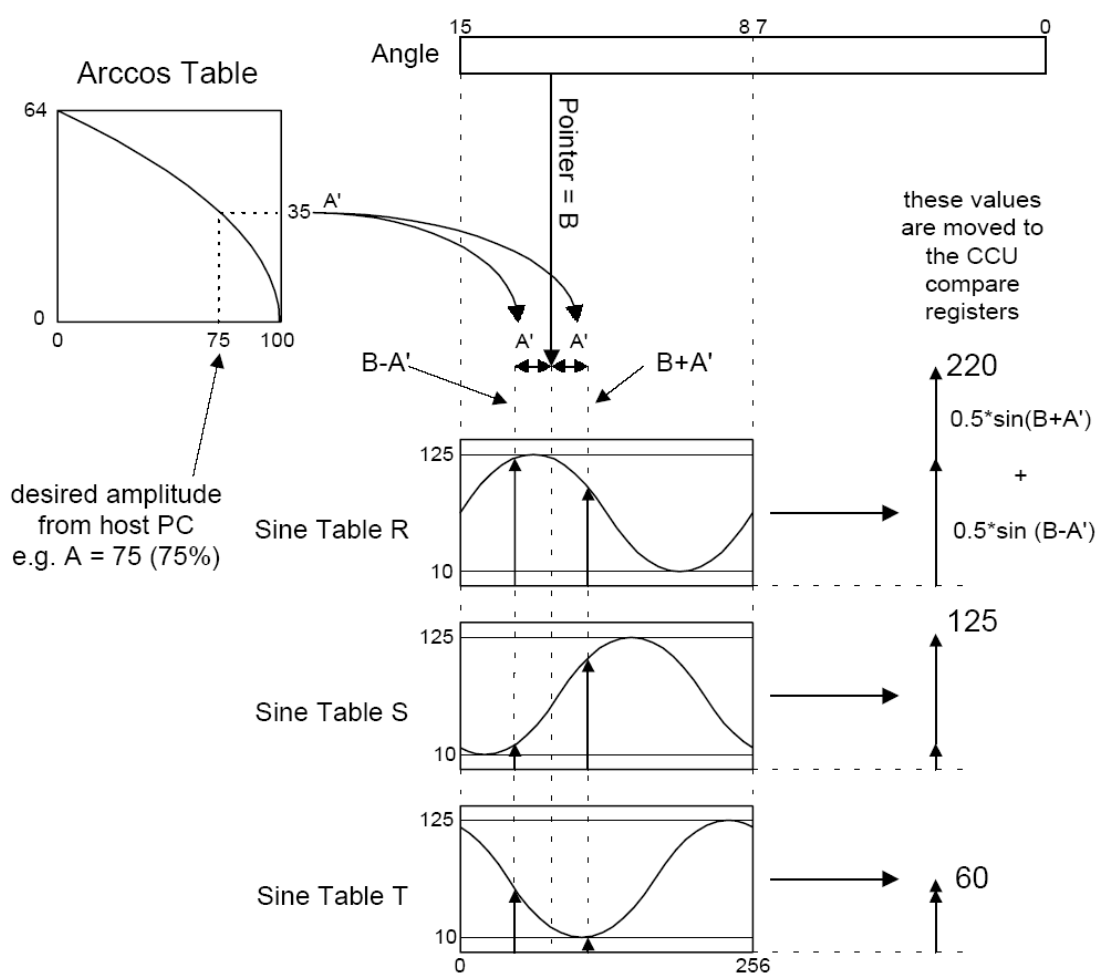
Obrázok 183, Bloková schéma skalárneho ASM pohonu s procesorom Infineon XC866



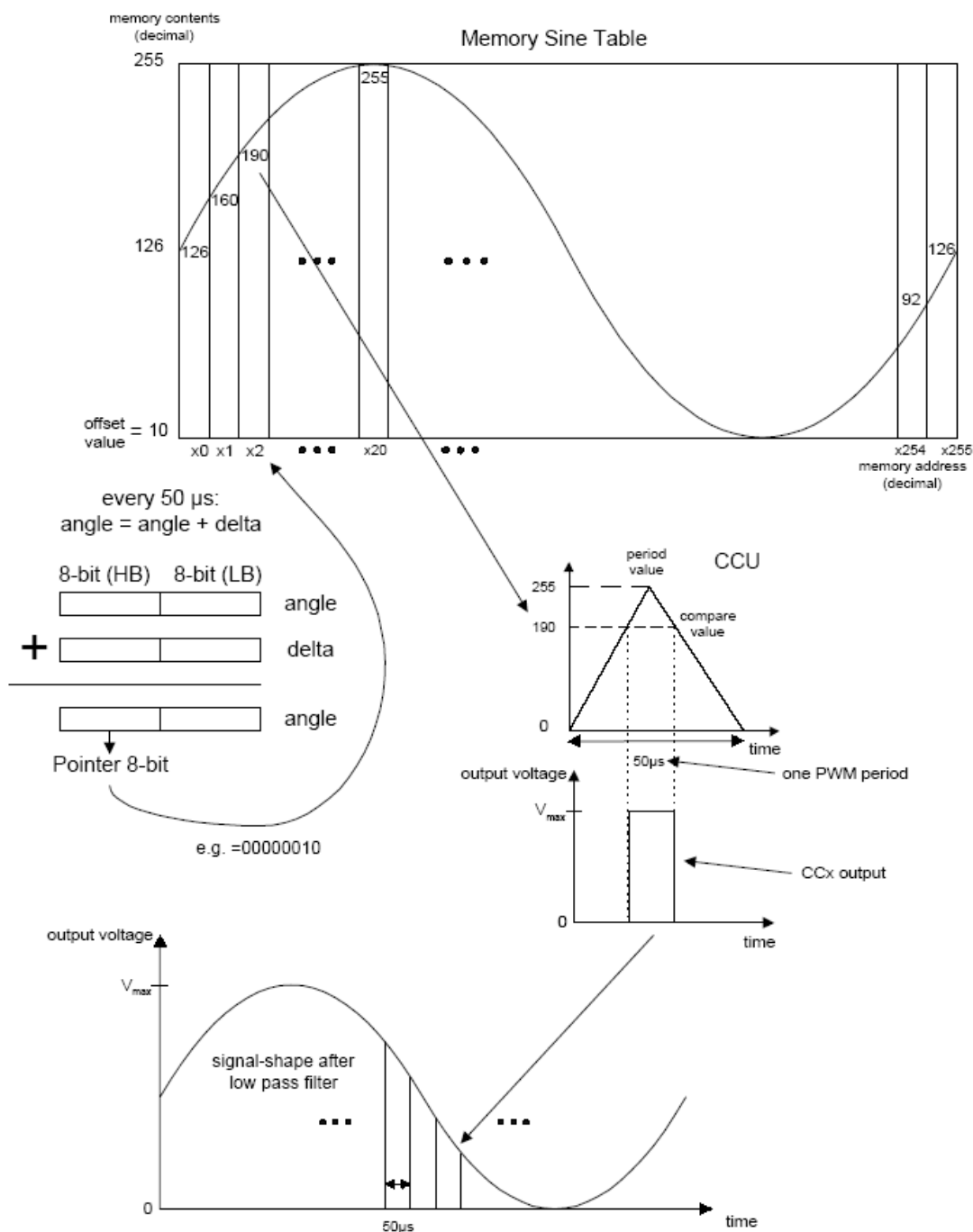
Obrázok 184, Výkonová časť 3-fázového striedača ASM



Obrázok 185, Princíp zmeny napätia na asynchrónnom pohone pomocou tabuľky



Vyššie uvedený spôsob umožňuje značne urýchliť výpočet napätia pre jednotlivé fázy tým, že mikroprocesor už má vopred vypočítané hodnoty sínusových vzoriek a časovo náročné násobenie je nahradené jednoduchým súčtom dvoch 8 bitových čísel pomocou ďalšej tabuľky. Obrázok 186, Princíp tvorby výstupného sínusového napätia s tabuľkou v pamäti C504



Príklad výpočtu pohonu:

Frekvencia PWM je v tomto prípade:

$$f_{PWM} = 20 \cdot 10^3 \text{ Hz}$$

Časovač T1 musí teda byť schopný generovať prerušenie každých:

$$T_{PWM} = \frac{1}{f_{PWM}} = \frac{1}{20 \cdot 10^3} = 50 \cdot 10^{-6} = 50 \mu s \quad (\text{s; Hz}) \quad (37.)$$

$$PERIODE = \frac{f_{OSC}}{\text{prescaler} \cdot f_{PWM} \cdot 2} = \frac{40 \cdot 10^6}{4 \cdot 20 \cdot 10^3 \cdot 2} = 250 \quad (1; \text{Hz, Hz}) \quad (38.)$$

Pre zabezpečenie tzv. „dead time“ je potrebné dodržať minimálne $t_{dead} = 1 \mu s$

$$OFFSET = \frac{t_{dead} \cdot f_{OSC}}{\text{prescaler}} = \frac{1 \cdot 10^{-6} \cdot 40 \cdot 10^6}{4} = 10 \quad (1; \text{s, Hz, 1}) \quad (39.)$$

Minimálnu a maximálnu frekvenciu je možné vyjadriť nasledujúcim vzťahom:

$$f_{MIN} = \frac{1}{65536 \cdot T_{PWM}} = \frac{1}{65536 \cdot 50 \cdot 10^{-6}} = 0.305 \text{ Hz} \quad (\text{Hz; s}) \quad (40.)$$

$$f_{MAX} = \frac{1}{256 \cdot T_{PWM}} = \frac{1}{256 \cdot 50 \cdot 10^{-6}} = 78.125 \text{ Hz} \quad (\text{Hz; s}) \quad (41.)$$

Vzťah potrebný pre zadanie požadovanej frekvencie asynchrónneho pohonu vyjadríme ako:

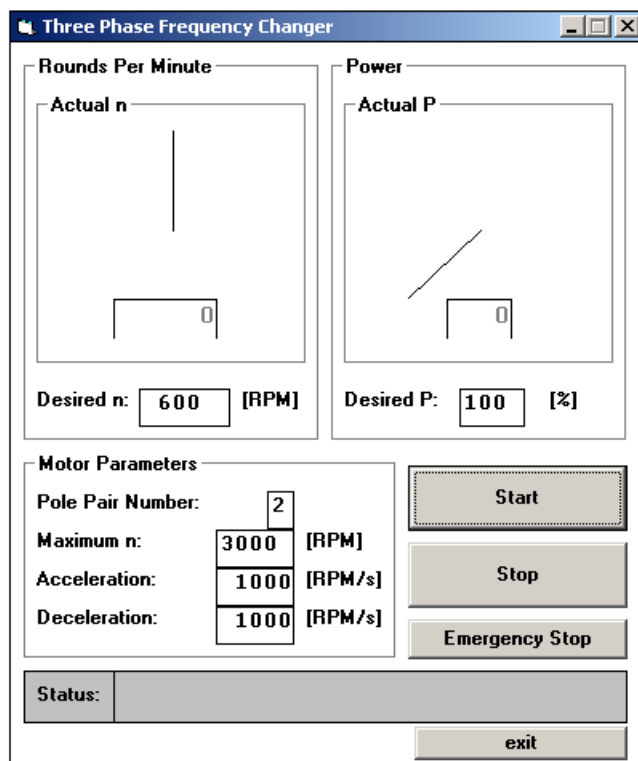
$$\text{angle} = \text{angle} + \text{delta} \quad (1; 1, 1) \quad (42.)$$

Kde *angle* je 16 bitová premenná a vyššia časť premennej obsahuje ukazovateľ do tabuľky vypočítaných hodnôt sínusových vzoriek a nižšia časť premennej *delta* znamená veľkosť- „rýchlosť“ prírastku pre výber hodnoty sínusu z tabuľky. Hodnota $\text{delta} \in (1 \div 256)$ zvyšuje obsah premennej *angle* o hodnotu *delta* každých $50 \mu s$.

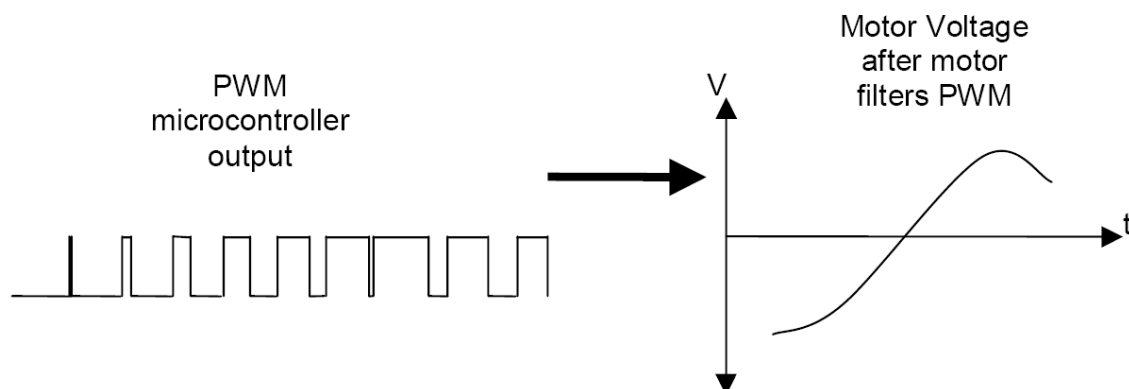
$$\text{delta}_{MAX_FREQUENCY} = \frac{f_{MAX}}{f_{PWM}} \cdot 65536 = 256 \quad (1; \text{Hz, Hz}) \quad (43.)$$

$$\text{delta}_{MIN_FREQUENCY} = \frac{f_{MAX}}{f_{PWM}} \cdot 256 = 1 \quad (1; \text{Hz, Hz}) \quad (44.)$$

Obrázok 187, Ovládací software pre pohon asynchrónneho motora

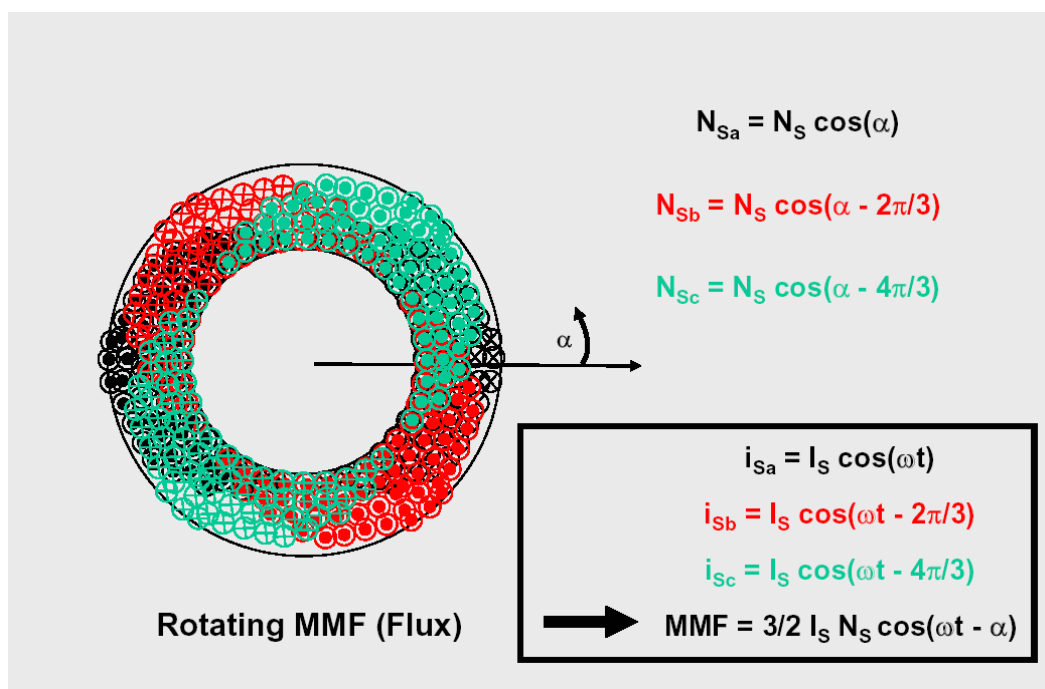


Obrázok 188, Napätie generované metódou SWPWM¹⁴³.

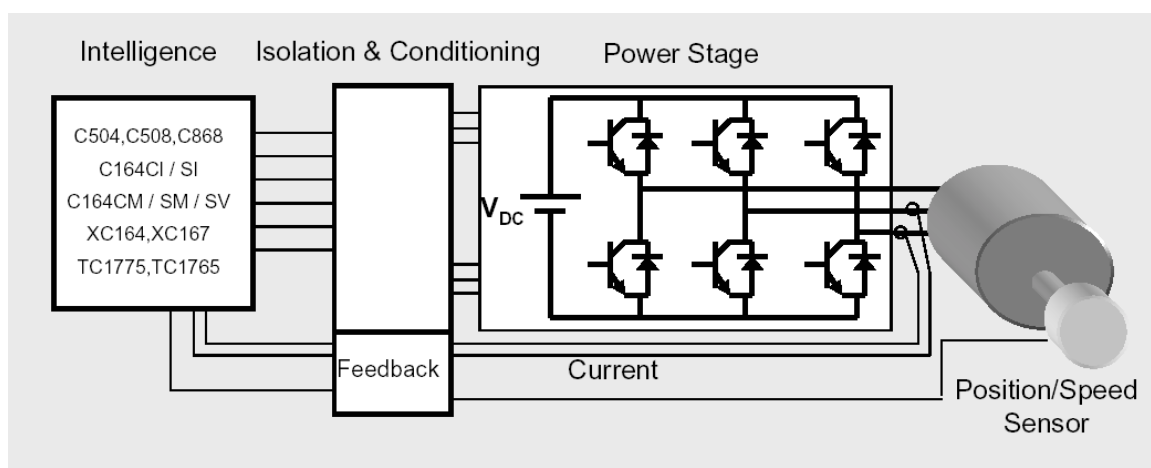


¹⁴³ Sinusoidal Weight Pulse Width Modulation

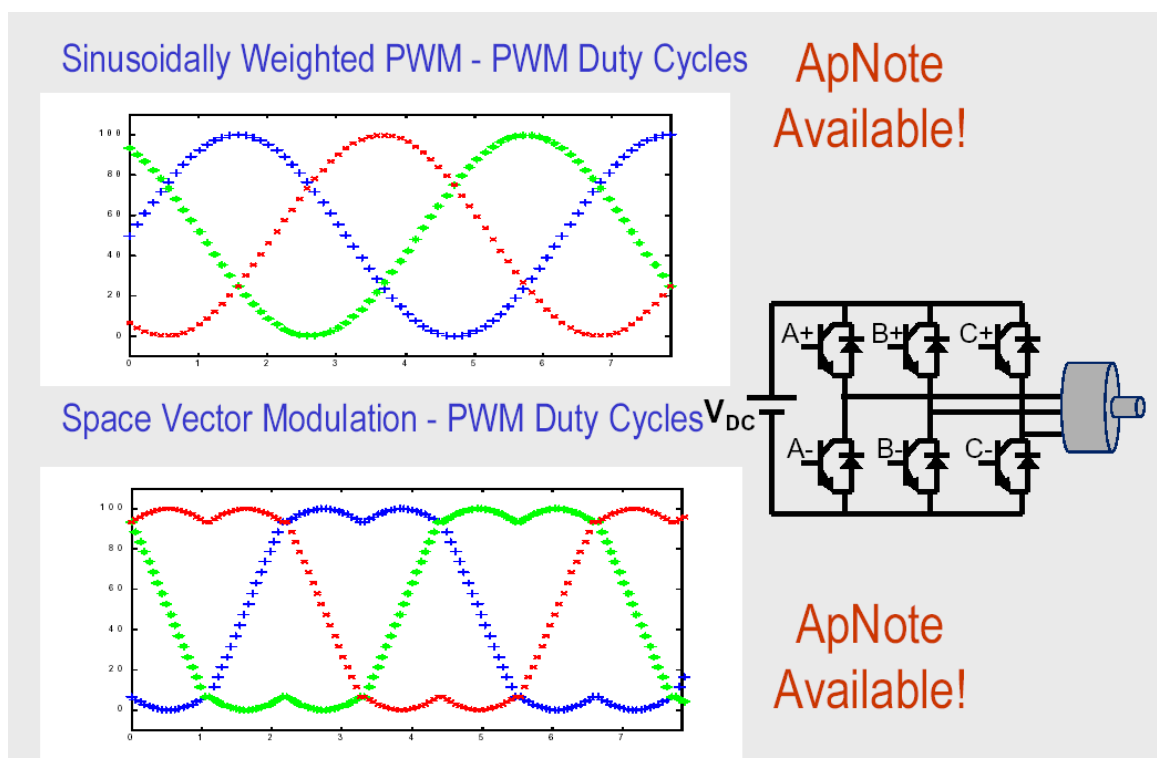
Obrázok 189, Vytvorenie rotujúceho magnetického poľa statora ASM



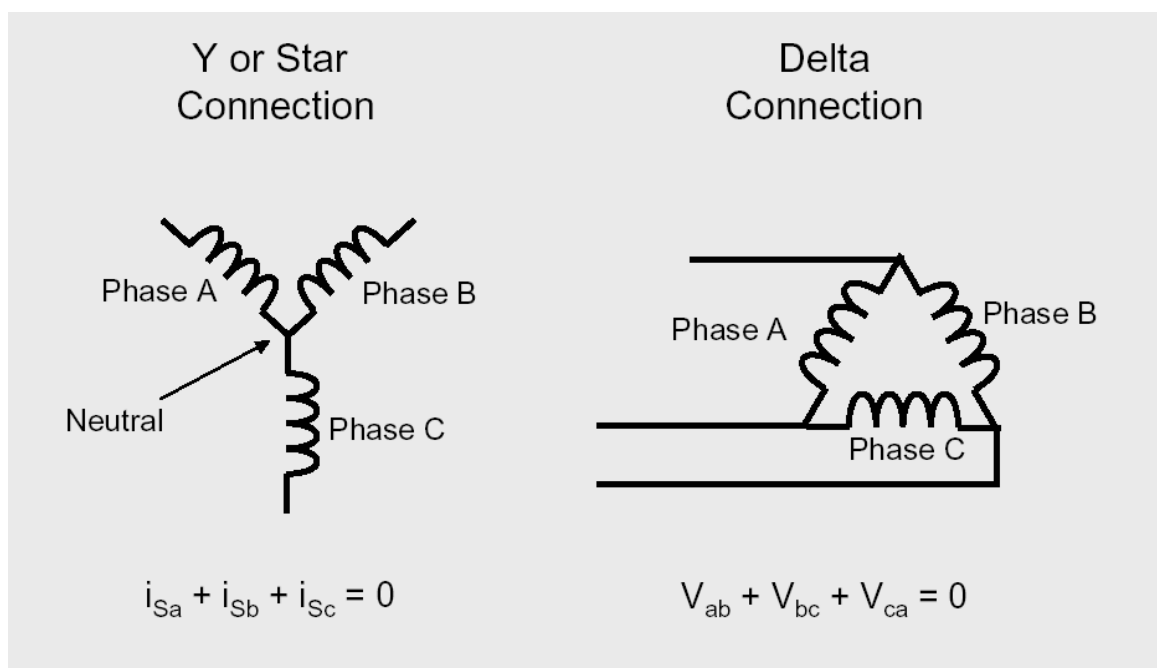
Obrázok 190, Schéma riadenia asynchrónneho motora s uzavretou regulačnou slučkou



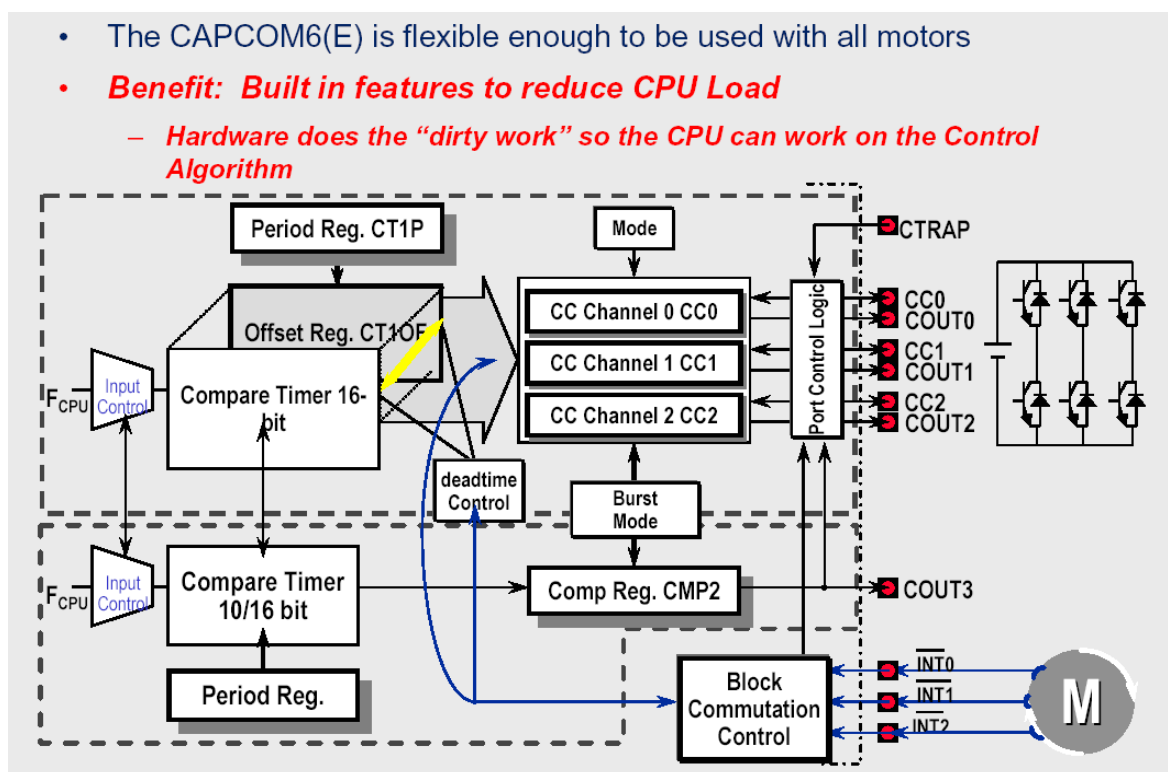
Obrázok 191, Principiálne znázornenie tvorby viacfázovej sústavy



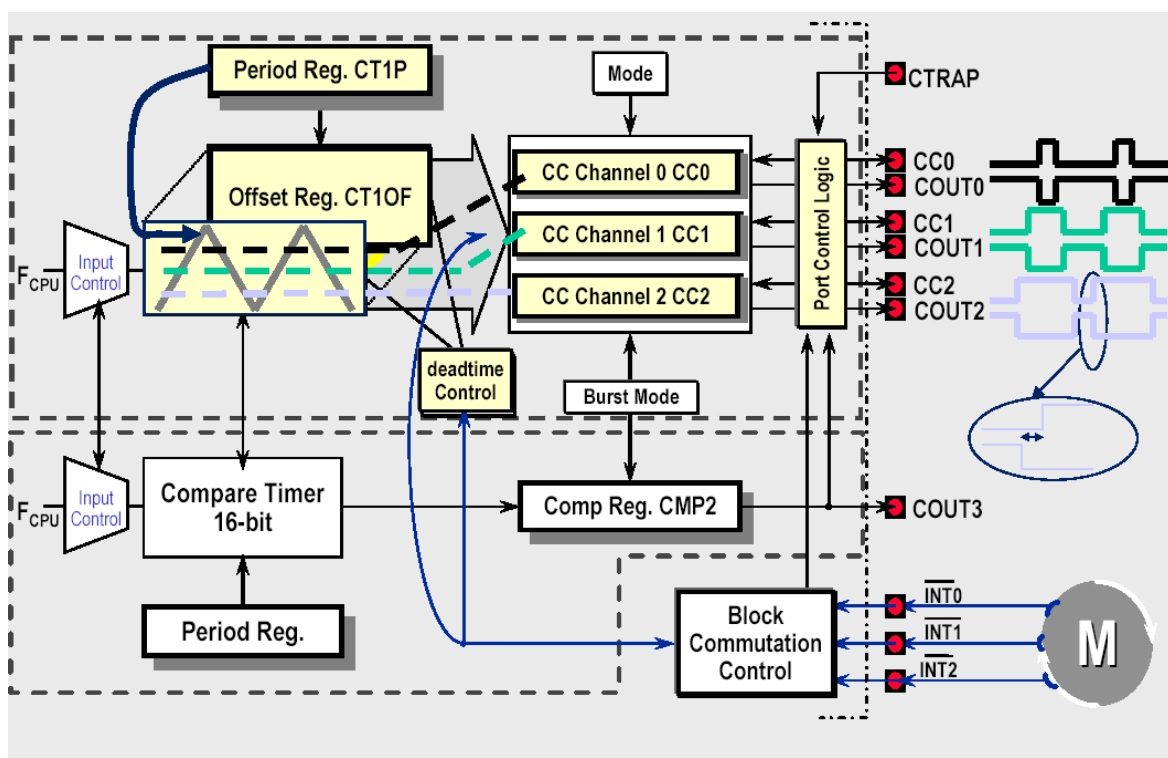
Obrázok 192, Základné zapojenie statora asynchrónneho motora



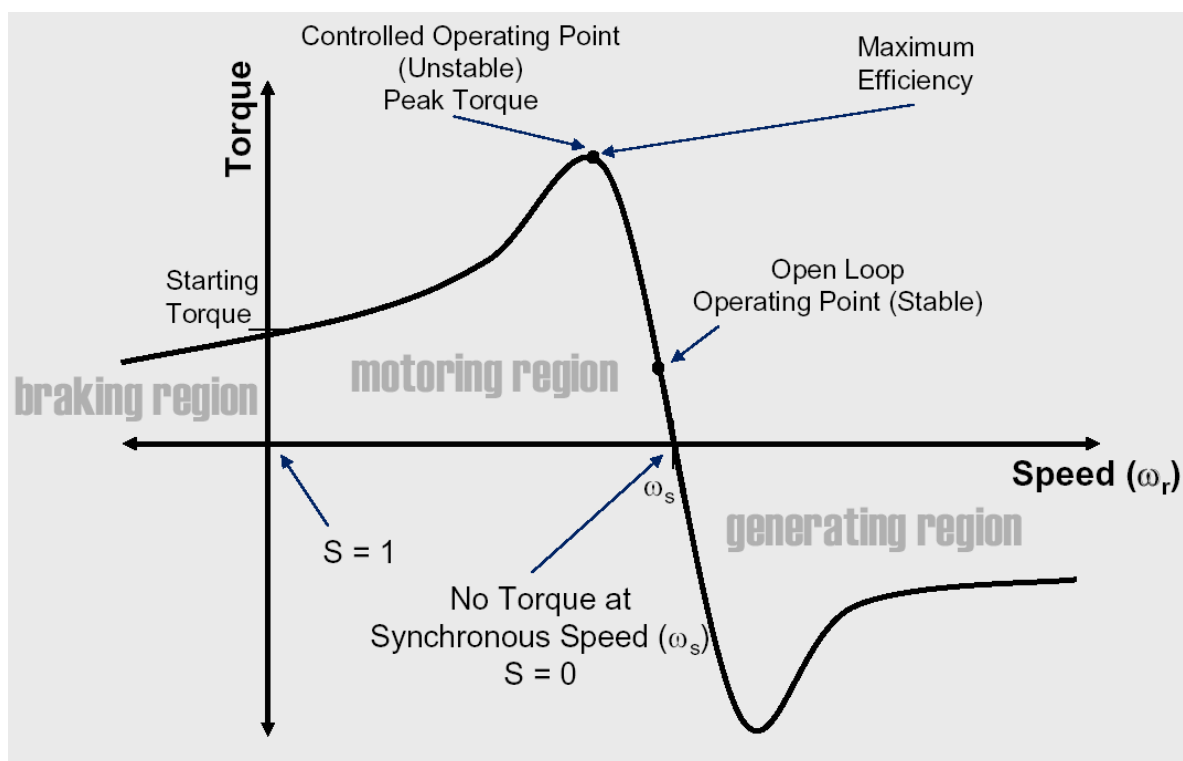
Obrázok 193, Riadenie asynchrónneho motora s SWPWM (Sinusoidal Weight PWM)



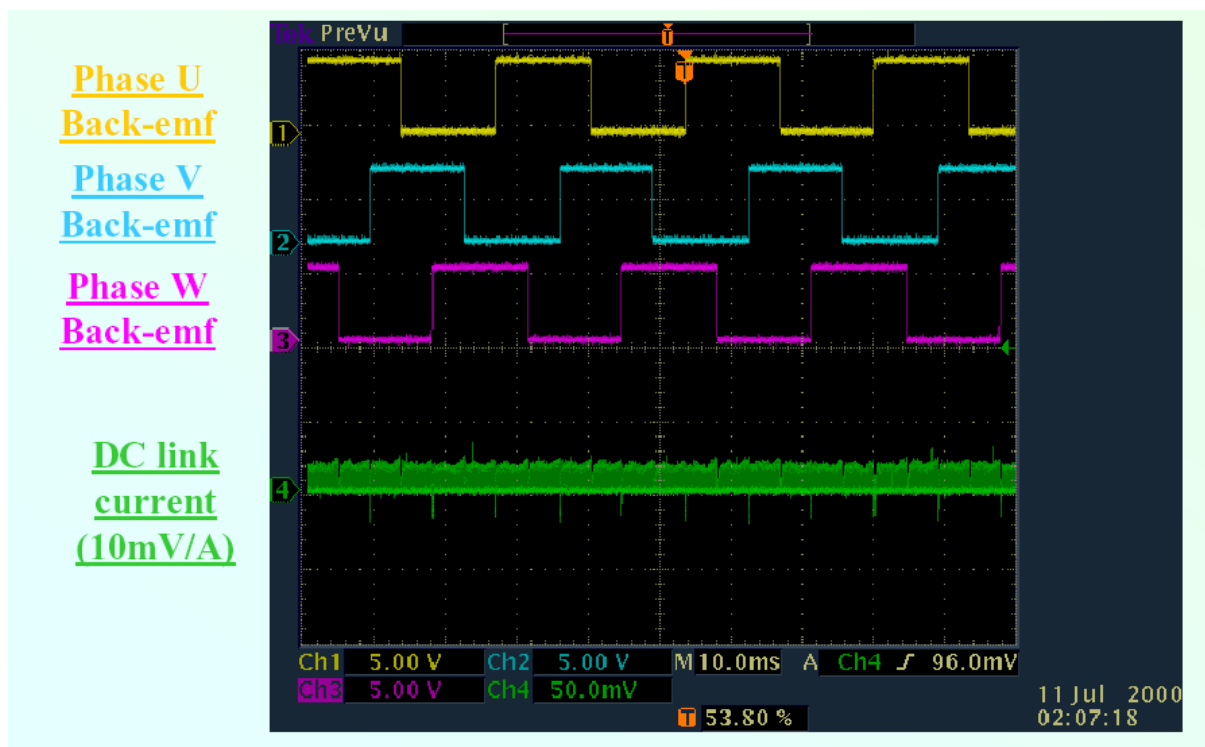
Obrázok 194, Riadenie asynchrónneho motora pomocou SVM (Space Vector Modulation)



Obrázok 195, Momentová charakteristika asynchrónneho motora



Obrázok 196, Namerané priebehy napätia na porte mikroprocesora C504



Zápis programu:

C:\Omega\data\Dropbox\Záloha\C51\ASMmotor\ASMMotor.c

```
1  #include <c:\keil\c51\inc\infineon\reg504.h>
2  #include <c:\keil\c51\inc\stdio.h> /* standard I/O .h-file */
3  #include <c:\keil\c51\inc\ctype.h> /* character functions */
4  #include <c:\keil\c51\inc\string.h> /* string and memory functions */
5  #include <c:\keil\c51\inc\stdlib.h>
6  #include <c:\keil\c51\inc\rtx51tny.h>
7  #include <c:\keil\c51\inc\absacc.h>
8  #include <c:\keil\c51\inc\math.h>
9
10 #define Fastx
11
12 #define PORT1 P1
13 #define PORT2 P2
14
15 enum Switch {ON=0, OFF=1};
16
17 sbit Tlac = P3^0;
18
19 void Pwm(void)
20 {
21 }
22
23 void Init(void)
24 {
25     #define XTAL 40E6
26     TH1=TL1=256-(unsigned char)((long int)XTAL/(long int)240000); //Generujem prerušenie kazdych
50us
27     TMOD|=0x20;
28     ET1=1;
29     PT1=1;
30     EA=1;
31 }
32
33 #define frek_pwm 20000
34 #define Krok 50.0E-6
35 xdata int Uac,Umax=255;
36 xdata float t,w,u_float,n;
37 xdata unsigned char f,p,U;
38 unsigned int i,Pocet_Vzoriek;
39 #ifdef Fast
40 xdata unsigned char x[10000];
41 #else
42 #include <c:\keil\c51\examples\asm_motor\init_mem.c>
43 void *ptr;
44 #endif
45
46 void Calcul(void)
47 {
48     #define MEM_BEGIN 0x1000
49     #define MEM_END 0x8000
50     i=0;
51     t=0;
52     w=2*3.14*f;
53     Uac=(Umax/50.0)*f;
54     n=60.*f/2;
55     if(Uac>=Umax) Uac=Umax;
56     Pocet_Vzoriek=frek_pwm/(2*f);
57     init_mempool(&XBYTE[MEM_BEGIN],MEM_END-MEM_BEGIN); //Inicializujem dostupnu pamat pre vzorky
sinusovky deklarovane v subore INIT_MEM.C
58     #ifndef Fast
59     if(ptr!=NULL) free(ptr);
60     ptr=malloc(Pocet_Vzoriek);
61     #endif
62     for(i=0x00; i<Pocet_Vzoriek; i++)
63     {
64         u_float=Uac*sin(w*t);
65         U=(unsigned char)u_float;
66         t=t+Krok;
67         #ifdef Fast
68         x[i]=U;
69         #else
70         ((unsigned char *)ptr)[i]=U;
71         #endif
72     }
```

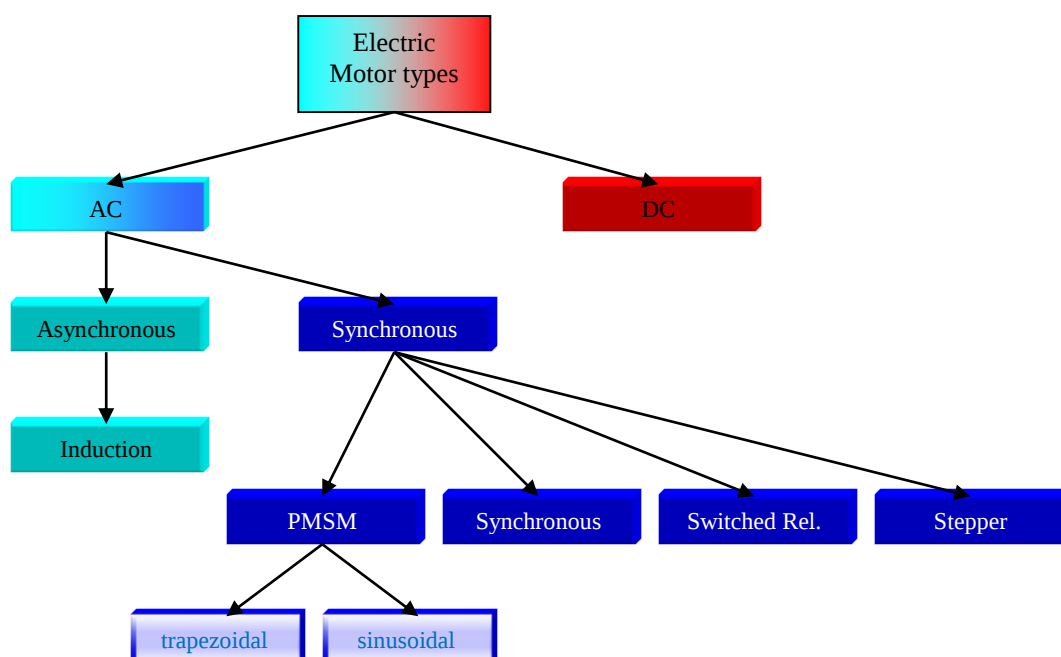
C:\Omega\data\Dropbox\Záloha\C51\ASM motor\ASM Motor.c

```
73     i=0;
74 }
75
76 bit  DIAG ;
77
78 void Int_T1 (void) interrupt 3 using 2
79 {
80     #ifdef Fast
81     if (DIAG == ON) PORT1=x[i++]; else PORT2=x[i++];
82     #else
83     if (DIAG == ON) PORT1=((unsigned char *) ptr)[i++]; else PORT2=((unsigned char *) ptr)[i++];
84     #endif
85     if (i==Pocet_Vzoriek)
86     {
87         i=0x00;
88         DIAG =~DIAG;
89         PORT1=PORT2=0x00;
90     }
91 }
92
93 void main (void)
94 {
95     Init ();
96     Pwm ();
97     f=50.0;
98     Uac=255;
99     Calcul ();
100    TR1=1;
101    while (1)
102    {
103        if (Tlac == 0)
104        {
105            TR1=0;
106            PORT1=PORT2=0x00;
107            Calcul ();
108            TR1=1;
109        }
110    }
111 }
112
113
```

17.10. Vektorové riadenie asynchrónneho motora

V automobilovom priemysle sú viacfázovými motormi vo väčšej miere nahrádzané komutátorové motory. Ako príklad môžeme uviesť synchrónny motor s permanentnými magnetmi (PMSM – Permanent Magnetic Synchronous Motors), ktoré sú nazývané ako bezkomutátorové synchrónne motory nazývané ako Brushless DC motors. Pre výborné vlastnosti ako vysoký požadovaný výkon a konštantný moment motora je predurčený do výkonovej elektroniky a moderných elektrických pohonov. Pre optimálne využitie týchto vlastností je používané vektorové riadenie – modulácia. Bližšie informácie (Vittek, 2004).

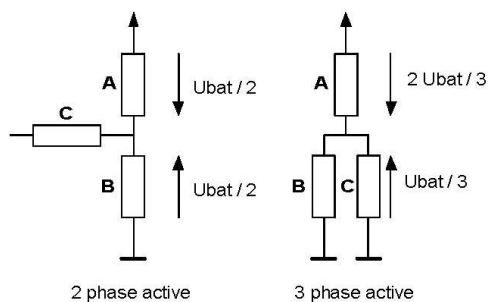
Obrázok 197, Rozdelenie elektrických pohonov



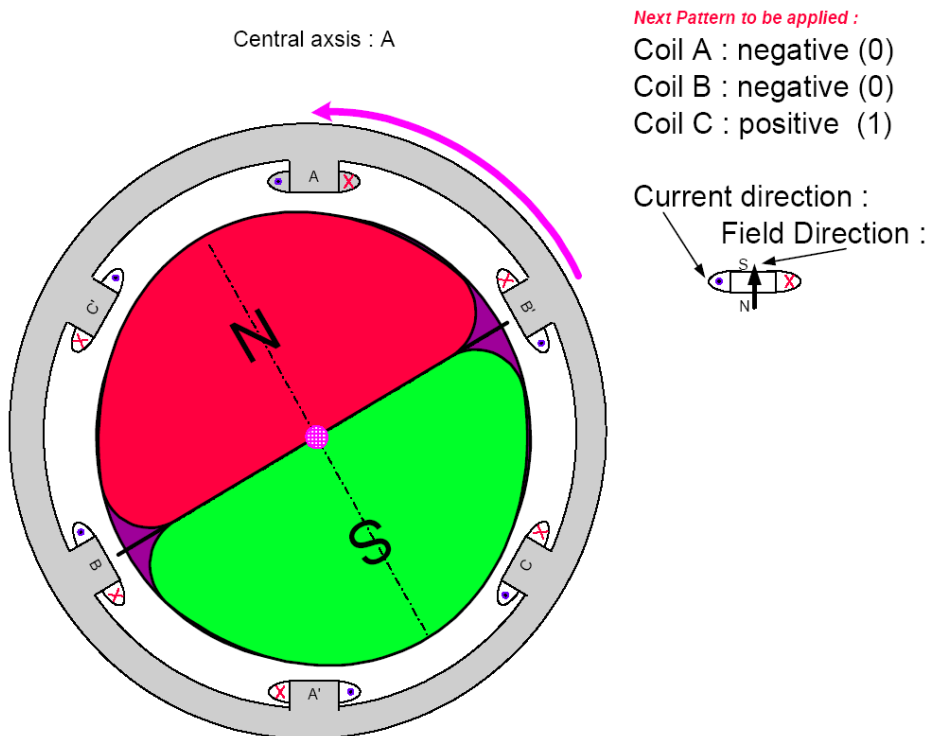
17.10.1. Výhody vektorového riadenia SVM (Space Vector Modulation)

- ✓ Vyššia dynamika riadenia
- ✓ Vysoká účinnosť dosahujúca až 86% oproti 75% pri klasickej SWPWM
- ✓ nižšie rýchlosti otáčania pri konštantnom momente
- ✓ konštantný moment pohonu
- ✓ lepší rozbeh pohonu s riadením momentu zvratu
- ✓ výborná dynamika pohonu s SVM

Obrázok 198, Základný rozdiel medzi SWPWM a SVM



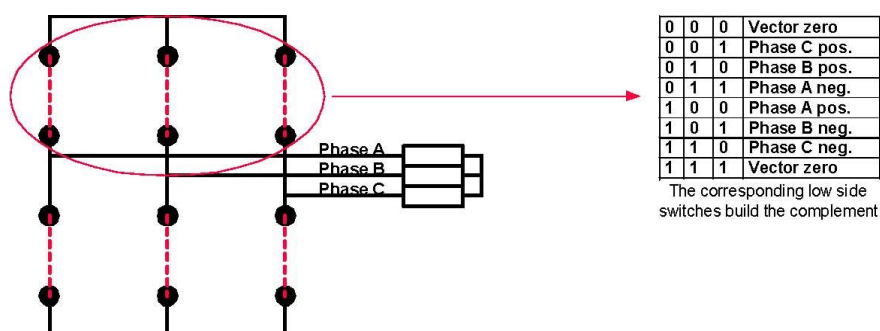
Obrázok 199, Detailné znázornenie magnetického obvodu PMSM s jedným pólom



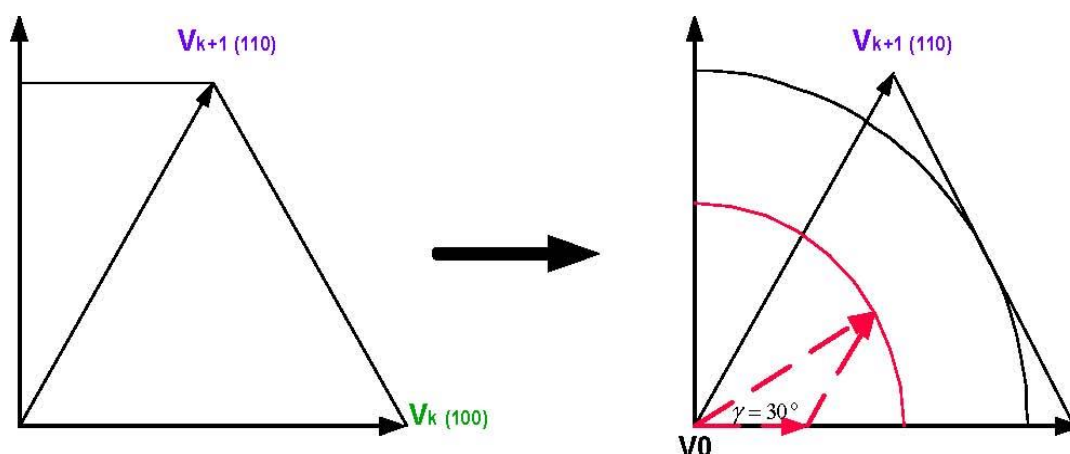
17.10.2. Principiálny opis SVM

Jednou z bežných metód reprezentácie SVM je model trojfázového modelu napájacieho zdroja, ktorý je schopný dodávať na výstupných svorkách kladné a záporné napätie. Obrázok 200 vysvetľuje, že môže nastať 8 možných stavov na spínacích prvkoch výkonového meniča. Každý stav z nich je reprezentovaný ako vektor. Tento spôsob umožňuje vytvoriť v rotujúci vektor napätia vo výkonovom meniči a môže sa nachádzať v niektorom zo 6 stavov. Zostávajúce 2 stavy sú nazývané ako nulové vektory (111, 000) a sú pripojené na GND, alebo Vcc. Nulové vektory sa nachádzajú vo vnútri geometrického útvaru šesťuholníka (hexagón) ako zobrazuje nasledujúci obrázok. Uhol medzi dvomi susednými sektormi je $\gamma=60^\circ$ elektrických.

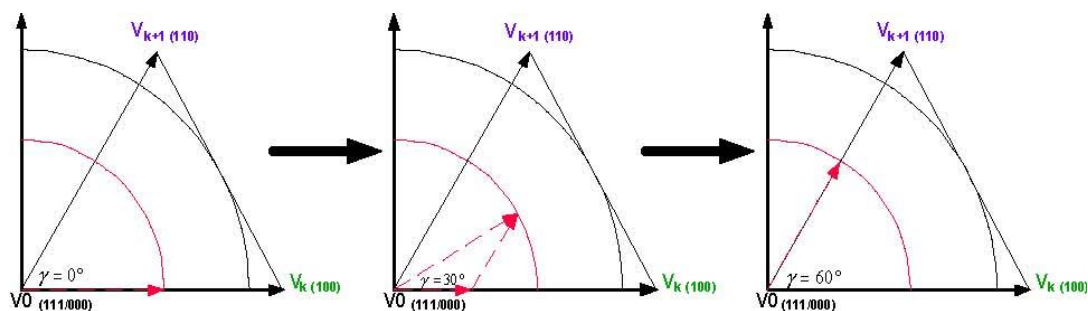
Obrázok 200, Spínací plán výkonového meniča s riadením SVM



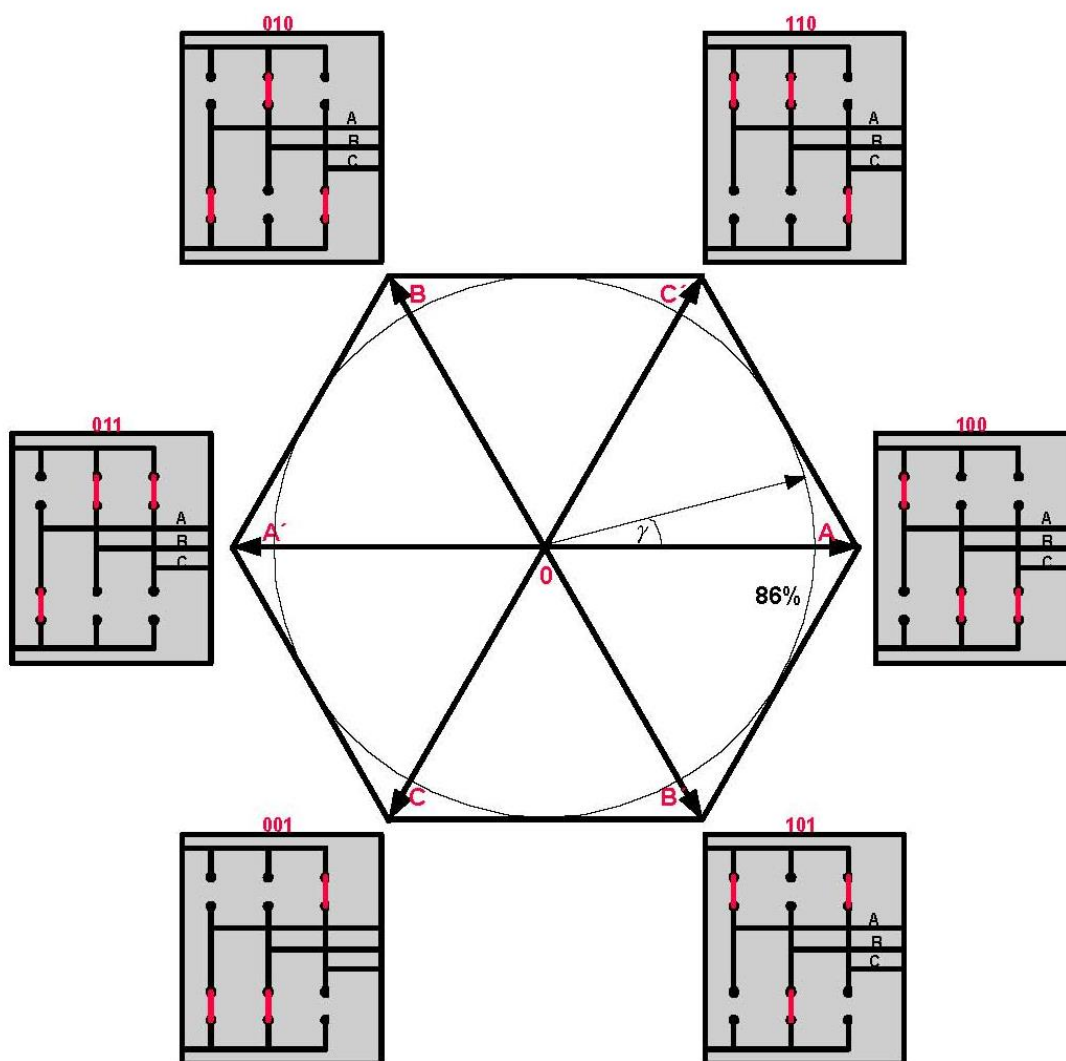
Obrázok 201, Znáznornenie rotujúceho vektora magnetického poľa pohonu s SVM



Obrázok 202, SVM modulácia pre γ ($0^\circ, 30^\circ, 60^\circ$)



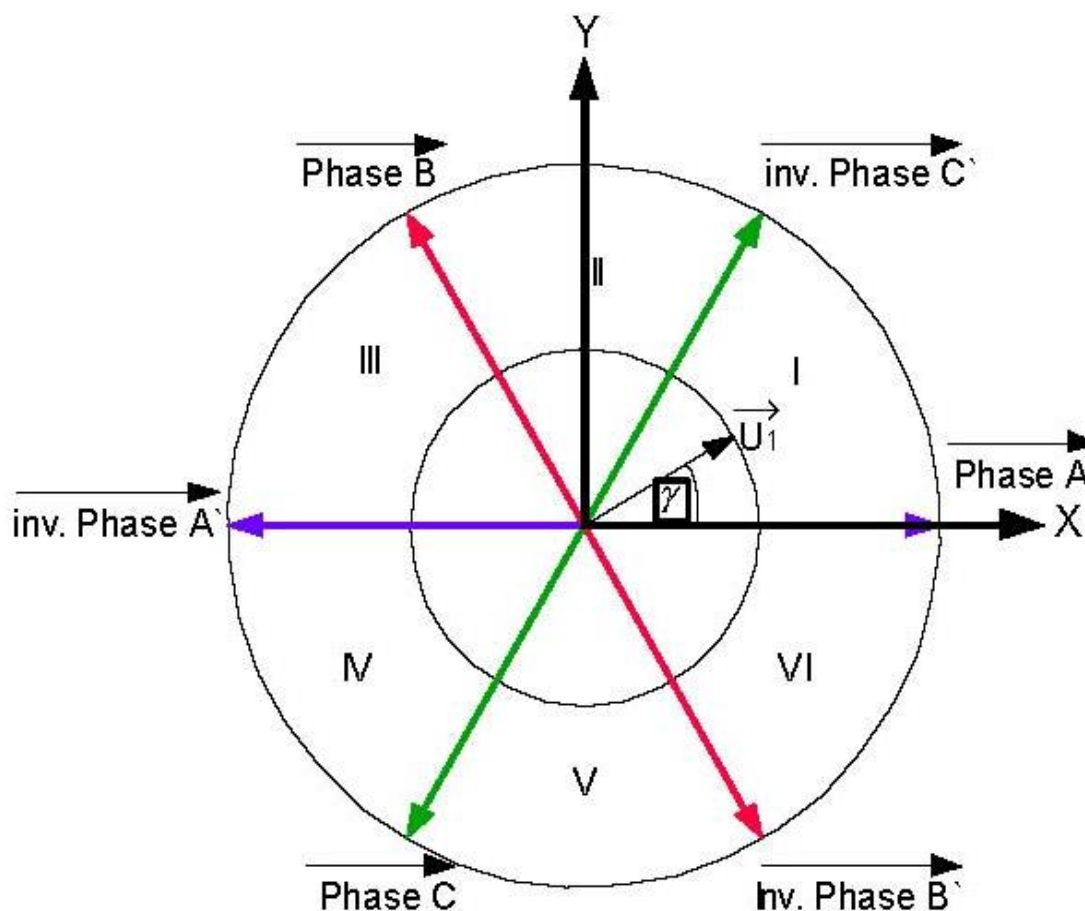
Obrázok 203, Vytvorenie rotujúceho vektora elektrického napätia pomocou SVM



17.10.3. Generovanie výstupného napätia pomocou SVM

SVM modulácia je založená na princípe, ktorý umiestňuje vektor napätia spínaním výkonových prvkov vo výkonovom meniči – invertore medzi 6 sektormi. Vektor napätia je teda reprezentovaný veľkosťou strednej hodnoty napätia dvoch susedných aktívnych veľkostí vektorov v priebehu jednej periódy PWM.

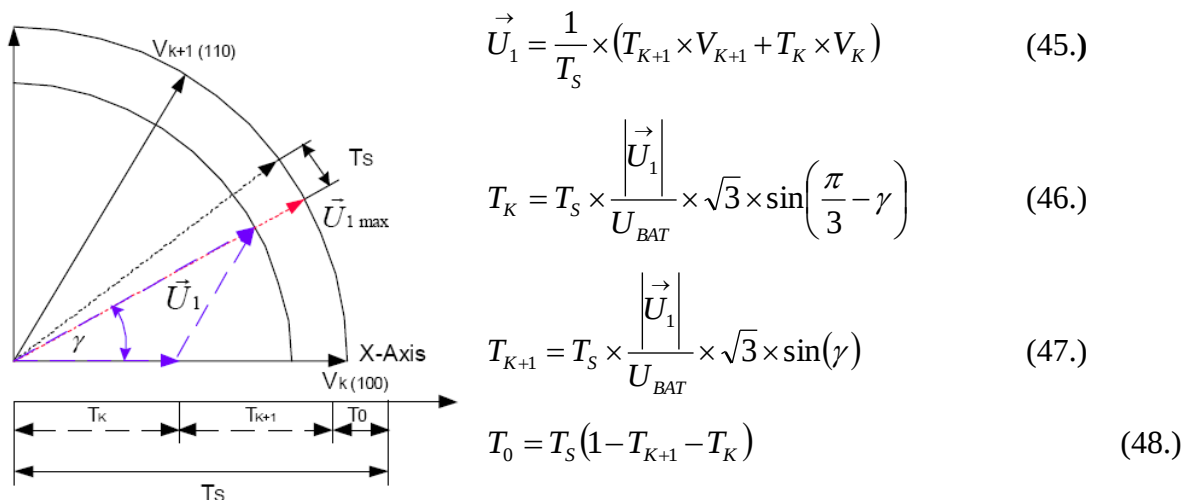
Obrázok 204, Sektory SVM znázorňujúce priebeh rotácie vektora



17.10.4. Výpočet vzoriek PWM pri SVM

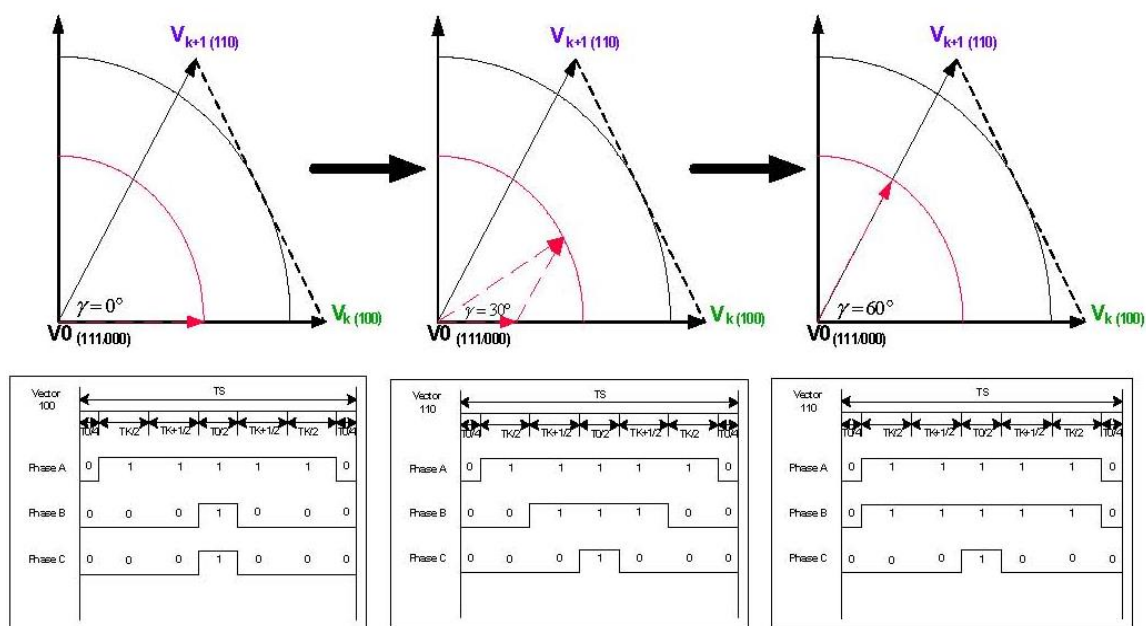
K vytvoreniu rotujúceho poľa vo vnútri kružnice šesťuholníka je použitá moderná technika ktorú nazývame SVM. Orientácia ľubovoľného úseku poľa je jednoznačne daná uhlom γ a veľkosťou U , ktorá je daná dĺžkou vektora. Každá pozícia vo vnútri sektorového trojuholníka je dosiahnutá kombináciou vektora V_K, V_{K+1} a nulového vektora T_0 . Všetky priestorové vektory sú realizované pomocou časového multiplexu všetkých troch vektorov V_K, V_{K+1}, T_0 počas vzorkovacieho intervalu T_s . Každá nasledujúca PWM sekvencia skladajúca sa z V_K, V_{K+1}, T_0 je uložená po uplynutí vzorkovacieho intervalu T_s , do CAPCOM6 registrov mikroprocesora. Vyššie uvedené znamená, že výber, alebo výpočet hodnôt musí byť ukončený za pred začatím ďalšieho vzorkovacieho intervalu T_s .

Obrázok 205, Napätový vektor pre SVM

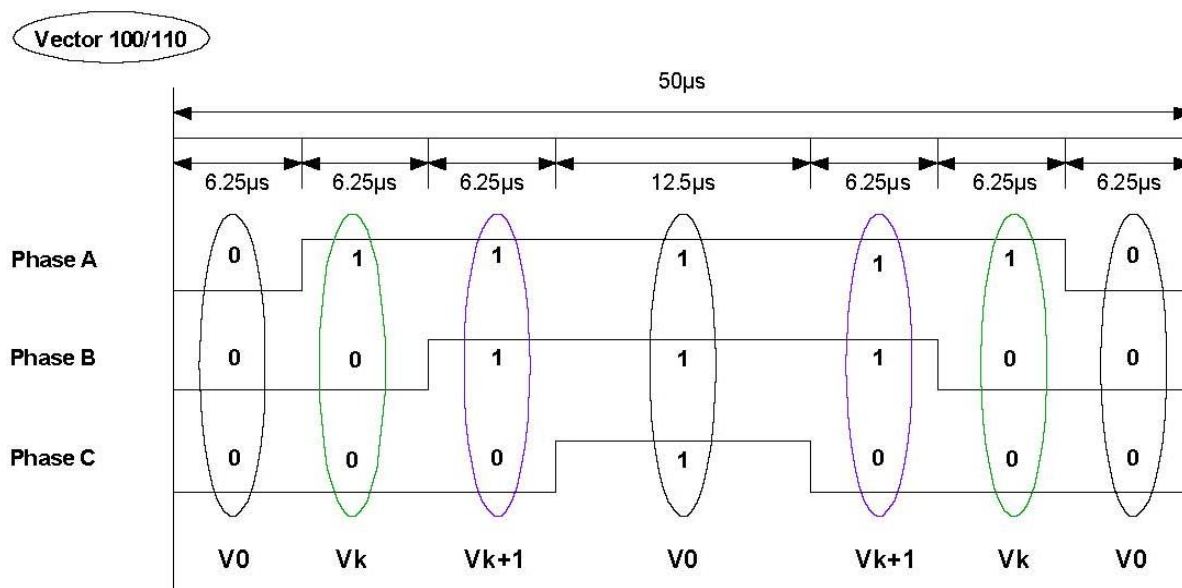


K získaniu takých vlastností SVM, akou je optimálny výkon a minimálna spínacia frekvencia prvkov výkonového meniča pre každé zariadenie, musí byť spínanie jednotlivých prvkov vykonávané tak, aby v ľubovoľnom okamihu bol len jeden v režime „invertor“. Tento stav nastane ak sekvencia spínania prvkov začína z nulového stavu [000] a invertorické spínacie prvky sú nastavené pokiaľ nie je dosiahnutý stav [111]. Obrázok 205 zobrazuje kompletný cyklus a jeho ukončenie, keď sa sekvencia vráti späť do nulového stavu.

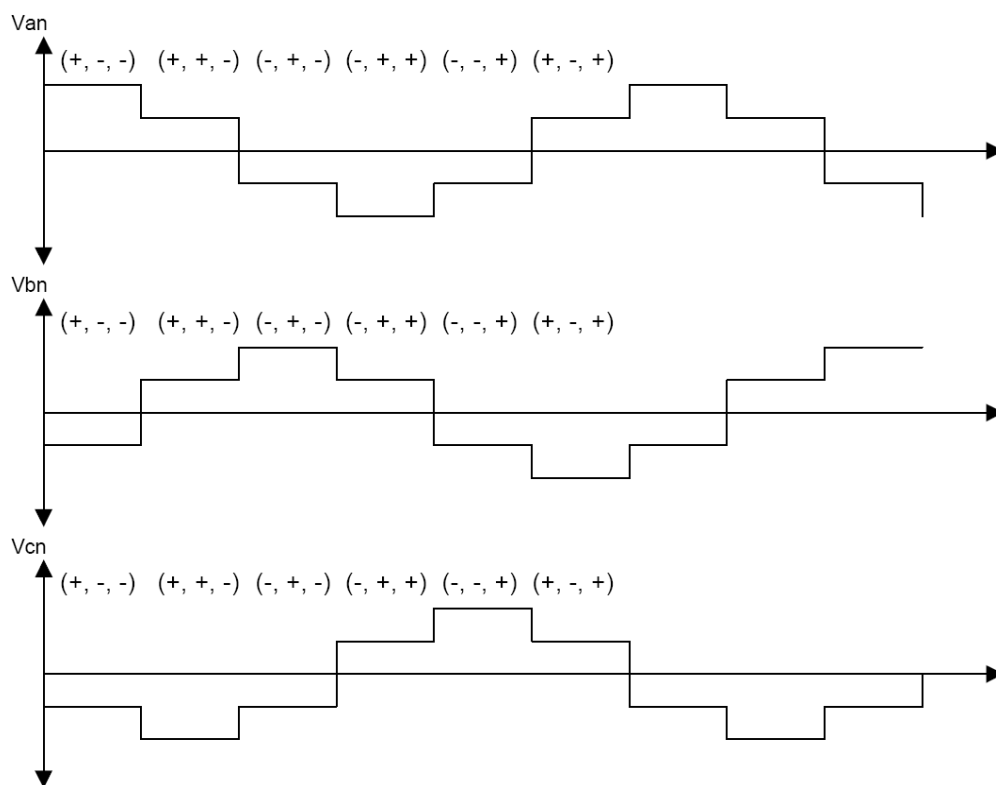
Obrázok 206, Priebehy napätí na spínacích prvkoch pri $\gamma=0^\circ, 30^\circ, 60^\circ$.



Obrázok 207, Priebehy napätí prvkoch pri $\gamma=30^\circ$ a vector length=50%, $f_{PWM}=20\text{kHz}$



Obrázok 208, Priebehy jednotlivých fázových napätí pri SVM modulácii.



Súčet všetkých fázových napätí V_{AN}, V_{BN}, V_{CN} v ľubovoľnom časovom okamihu je rovný nule a je možné ho jednoducho transformovať pomocou nasledujúceho vzťahu na vektor $\underline{u}_S$:

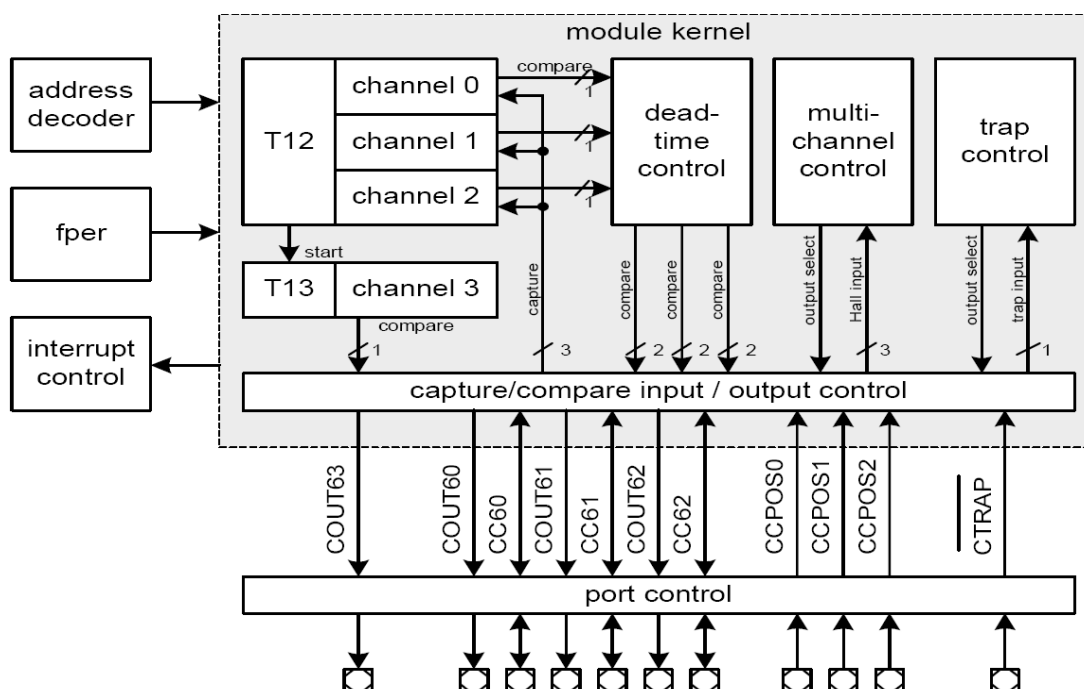
$$\underline{u}_S = V_{AN}(t) \cdot e^{j0} + V_{BN}(t) \cdot e^{j\frac{2}{3}\pi} + V_{CN}(t) \cdot e^{-j\frac{2}{3}\pi} \quad (49.)$$

Tabuľka 15, Vzájomná závislosť medzi uhlom, T_K , T_{K+1} , T_0 , α , β .

Degree(°)	TS(μs)	TK(μs)	TK+1(μs)	T0(μs)	PWM(%)	Alpha	Beta
0	50	43.30	0	6.69	100	0.866	0
10	50	38.30	8.68	3.02	100	0.766	0.174
20	50	32.14	17.10	0.76	100	0.643	0.342
30	50	25.00	25.00	0	100	0.5	0.5
40	50	17.10	32.14	0.76	100	0.342	0.643
50	50	8.68	38.30	3.02	100	0.174	0.766
60	50	0	43.30	6.69	100	0	0.866

Degree(°)	TS(μs)	TK(μs)	TK+1(μs)	T0(μs)	PWM(%)	Alpha	Beta
0	50	21.65	0	28.35	50	0.866	0
10	50	19.15	4.34	26.5	50	0.766	0.174
20	50	16.07	8.55	25.38	50	0.643	0.342
30	50	12.5	12.5	25.0	50	0.5	0.5
40	50	8.55	16.07	25.38	50	0.342	0.643
50	50	4.34	19.15	26.5	50	0.174	0.766
60	50	0	21.65	28.35	50	0	0.866

Obrázok 209, Detailné zobrazenie CAPCOM6 jednotky mikroprocesora C164 a C508.



Program a príslušnú dokumentáciu¹⁴⁴ je možné nájsť na stránkach www.infineon.com.

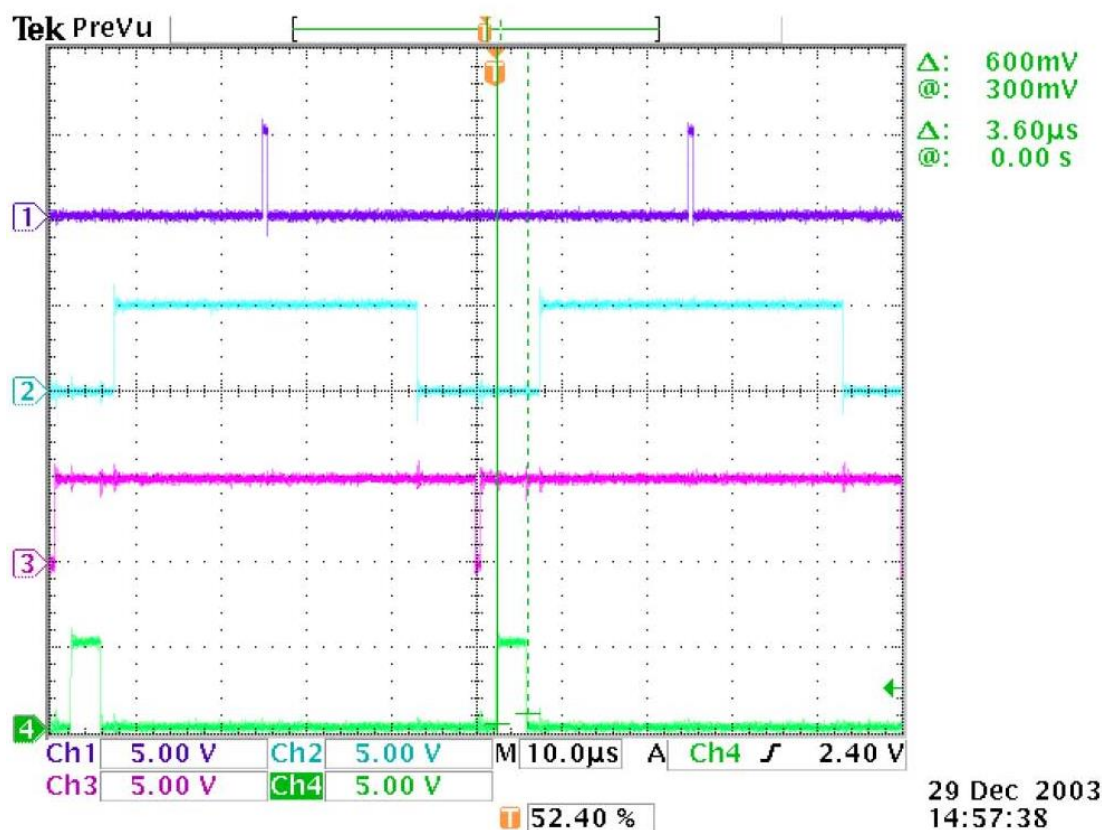
144

http://www.infineon.com/dgdl/Ap0802211_3-Phase-Currents.pdf?folderId=db3a304412b407950112b41a69152a86&fileId=db3a304412b407950112b41a69672a87

17.10.5. Analýza vytťaženia mikroprocesora C508 pri modulácii SVM

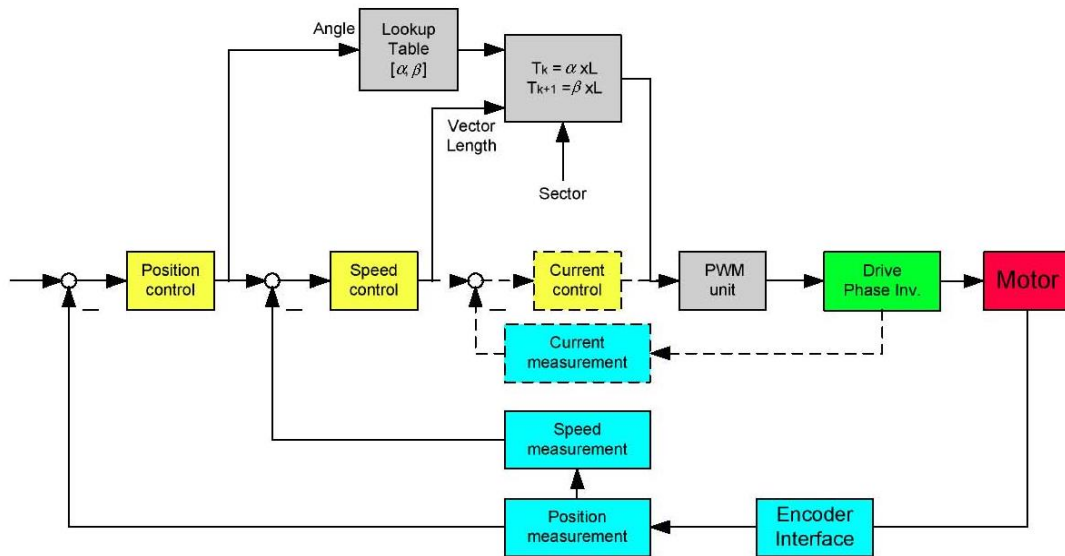
Obrázok 210 ukazuje koľko času potrebuje mikroprocesor C508 pre výpočet SVM. Je to približne $3,6\mu\text{s}$ pri frekvencii CPU 40MHz na uloženie nových obsahov CAP/COM registrov jednotky CAPCOM6 mikroprocesora.

Obrázok 210, Vytťaženie mikroprocesora (CPU Load) pri SVM

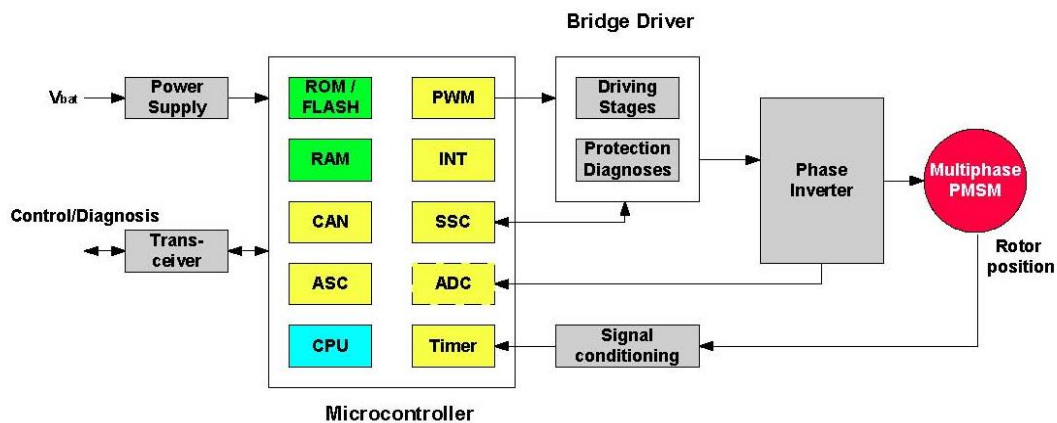


Poznámka: bližšie informácie¹⁴⁵ je možné získať na www.infineon.com.

Obrázok 211, Kaskádne riadenie viacfázového pohonu s SVM

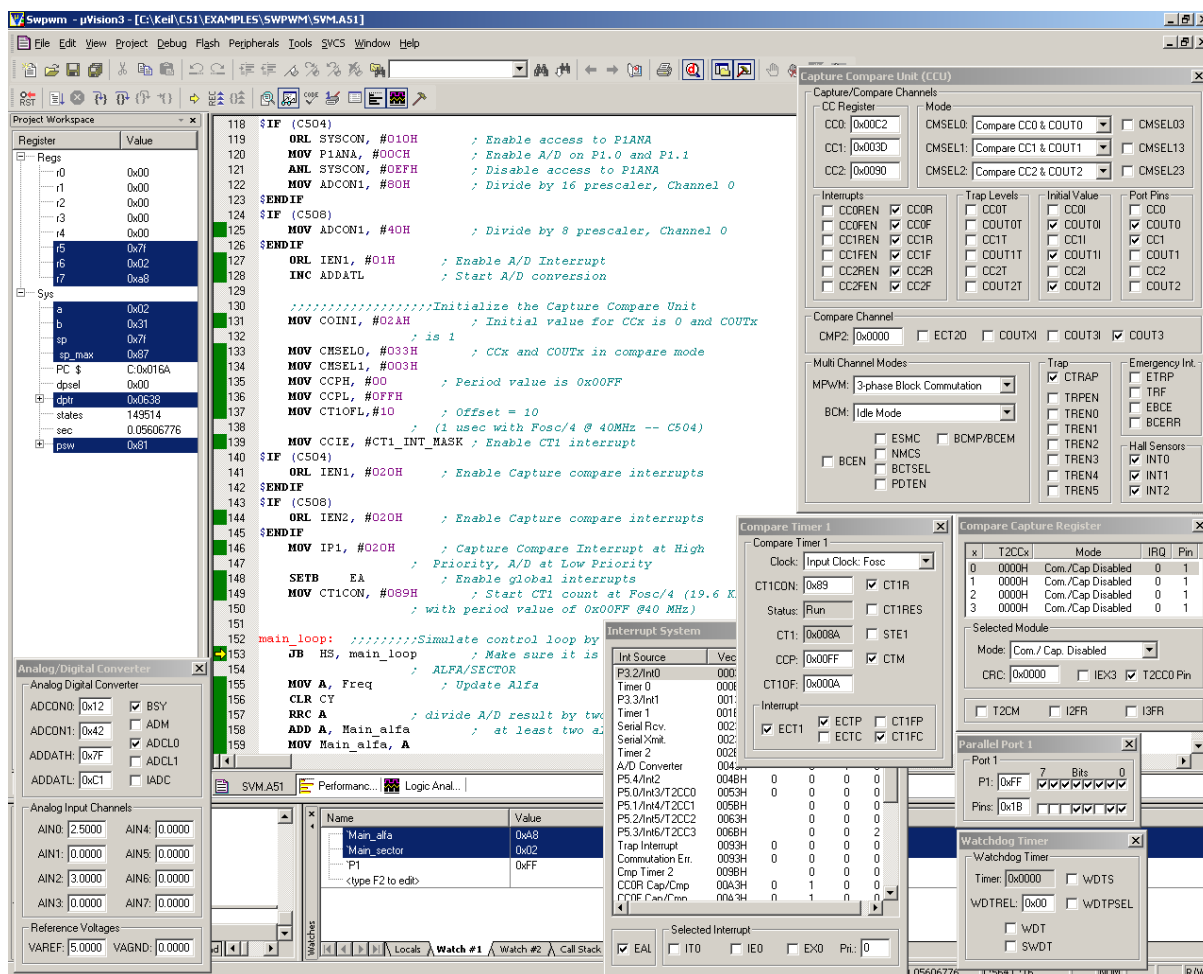


Obrázok 212, Principiálne riadenie viacfázového pohonu PMSM s SVM



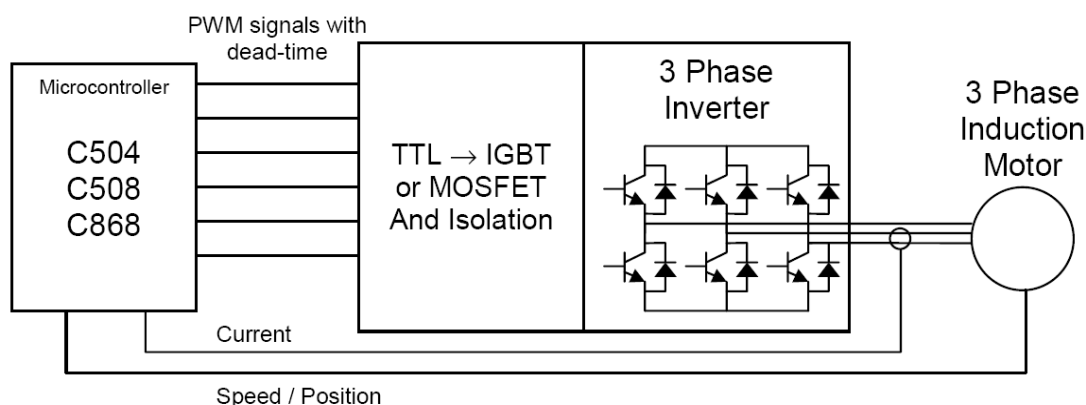
Poznámka: bližšie informácie¹⁴⁶ je možné získať na www.infineon.com.

Obrázok 213, Ladenie a simulácia programu v prostredí µVision3

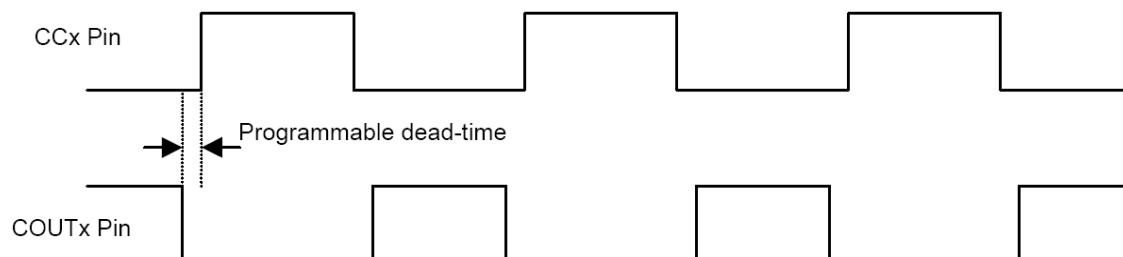


Príklad: S pomocou programovacieho jazyka A51, C51, vytvorte program, ktorý bude riadiť frekvenčný menič s IGBT ktorý napája jednosmerný motor s cudzím budením, jednofázový komutátorový motor a trojfázový asynchrónny motor (**ASM**) s klieťkovou kotvou. Pracovná frekvencia je 20kHz. Daný menič musí byť schopný napájať jednosmerným a striedavým napätím s premenlivou veľkosťou napätia a frekvencie daný pohon. Napájacie napätie pre motor bude v rozsahu 0V po 400V_{AC} s frekvenciou 0 až 150Hz. Výstupy portu mikroprocesora C508 budú s pomocou 20kHz PWM ovládať trojfázové mostové zapojenie s IGBT, ktorý plní funkciu napäťového striedača. Striedač bude ovládaný pomocou dvojice tlačidiel. Dve tlačidlá budú slúžiť na voľbu otáčok motora a zároveň na zmenu smeru otáčania. Zároveň program doplníte o ovládanie pomocou sériového rozhrania RS232 s nasledujúcimi parametrami: 2400b.s⁻¹, 8 dátových bitov, 1 štart a stop bit, bez parity. Výstupné napätie bude modulované pomocou SVM (Space Vector Modulation).

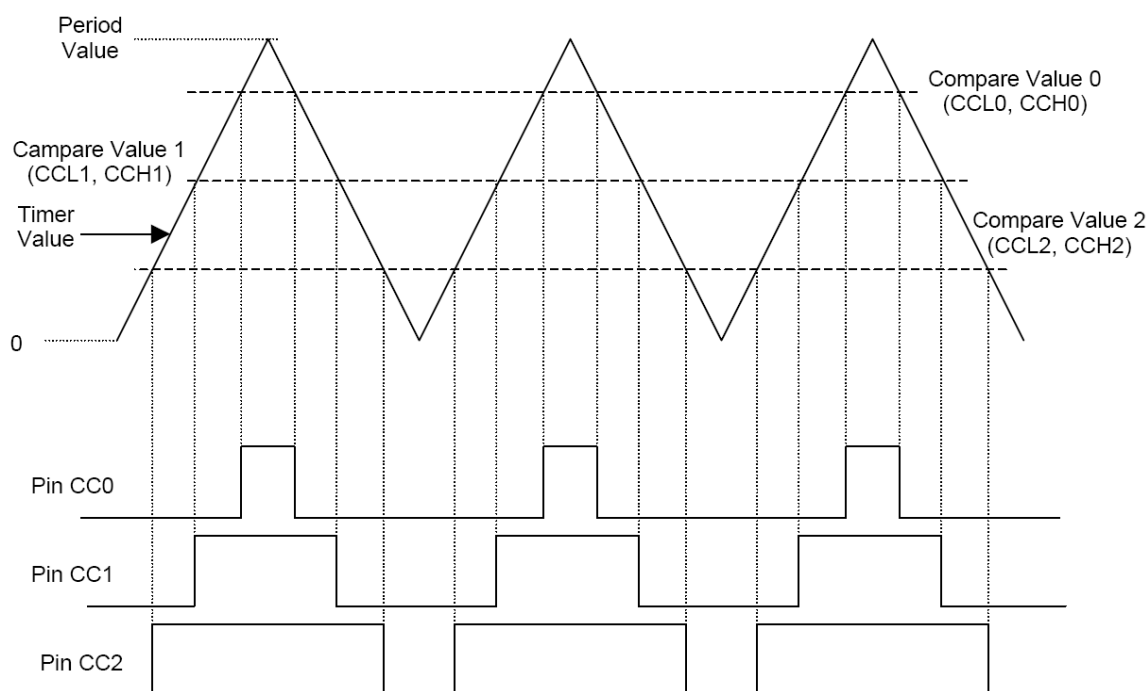
Obrázok 214, Schéma zapojenia pohonu s SVM (Space Vector Modulation)



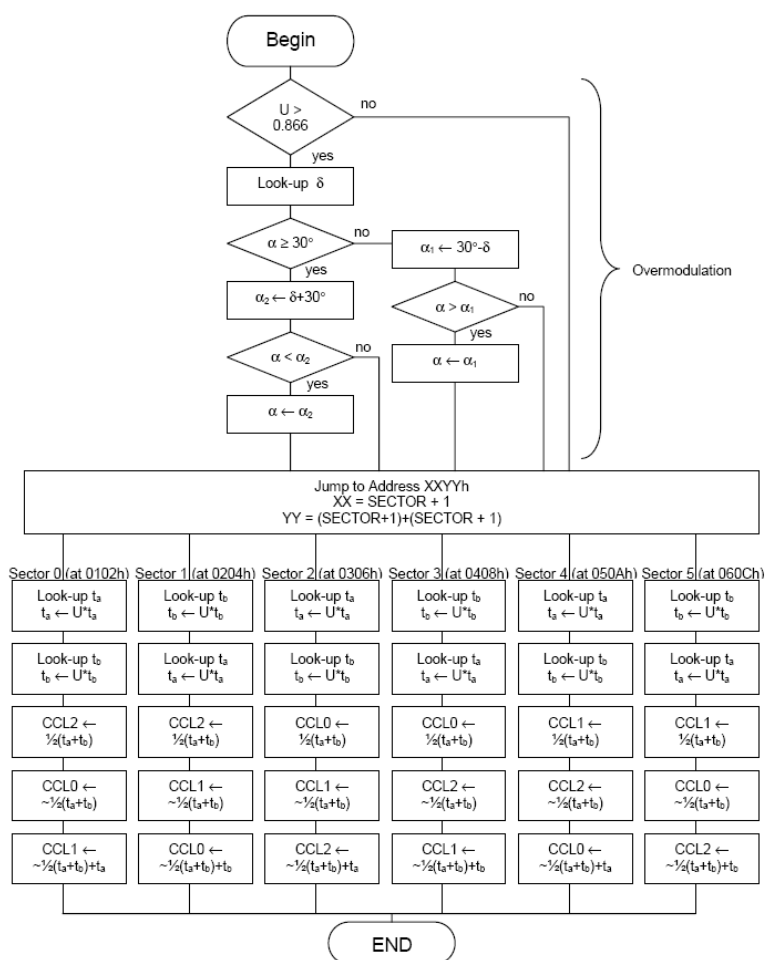
Obrázok 215, Výstup mikroprocesora CCx a COUTx s programovateľným dead-time



Obrázok 216, Generovanie napätia s CAPCOM6 v móde 1 pre SVM (Center PWM).



Obrázok 217, Vývojový diagram riadenia pohonu s SVM (Space Vector Modulation)



Obvyklý spôsob modulácie napätia SWPWM pre pohony spočíva vo vytvorení sústavy troch sínusových napätí s rovnakou amplitúdou a vzájomným fázovým posunom 120° elektrických. Ak generujeme napätie pomocou SVM, tak periodicky zvyšujeme uhol vektora pomocou programu v mikroprocesore, čo po dosadení do vzťahov uvedených nižšie rotáciu vektora napätia okolo vlastnej osi. Ak aproximujeme $\underline{u}_s$ použitím $\underline{u}_1$ a $\underline{u}_2$ vytvoreného pomocou t_a a t_b vytvoríme tak vektor napätia, ktorý môžeme umiestniť kdekoľvek v priestore. Obrázok 221 to názornejšie vysvetľuje.

Pri transformácii uvažujeme napätie priestorového vektora vyjadreného ako

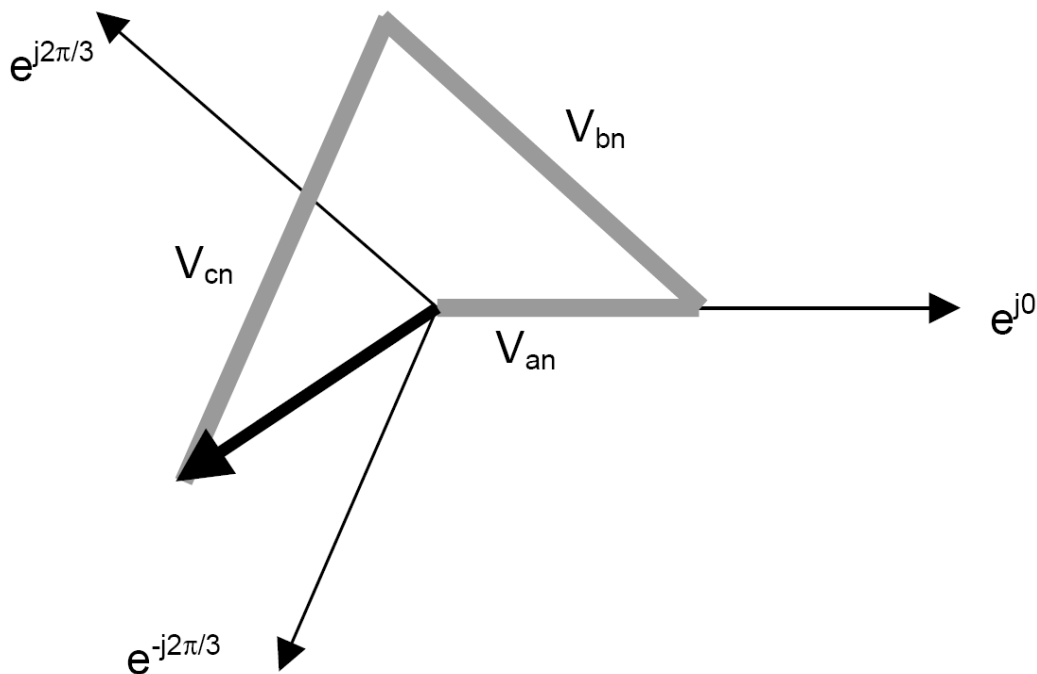
$$\underline{u}_s = t_a \cdot \underline{u}_1 + t_b \cdot \underline{u}_2 \quad (50.)$$

$$t_a = \frac{2 \cdot U}{\sqrt{3}} \cdot \sin \alpha \quad (51.)$$

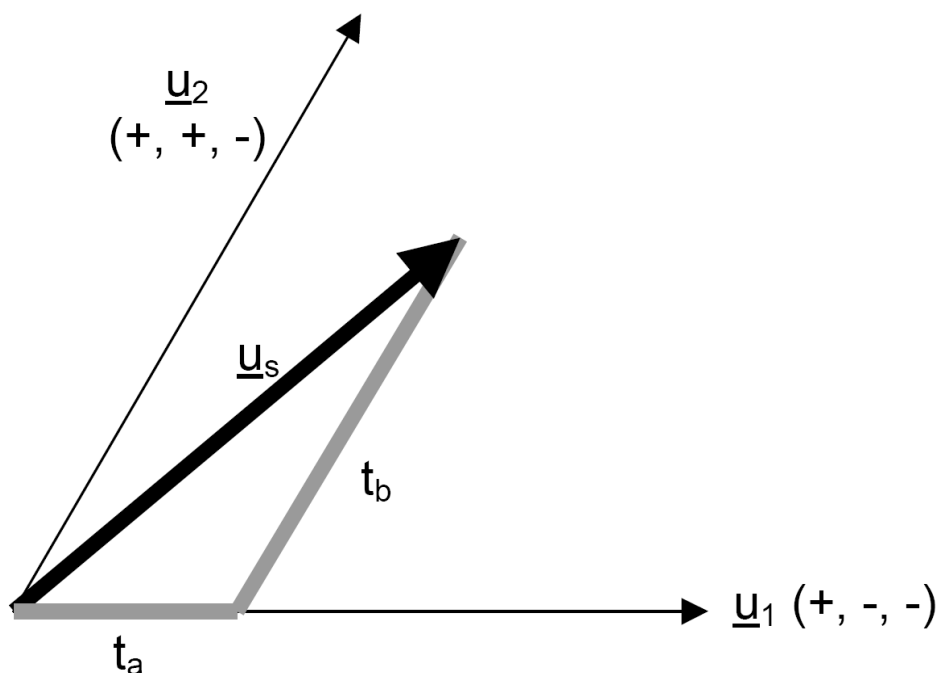
$$t_b = U \cdot \left[\cos(\alpha) - \frac{\sin(\alpha)}{\sqrt{3}} \right] \quad (52.)$$

kde $U = |\underline{u}_s|$ je tzv. Modulačný index, $\alpha = \angle \underline{u}_s$ je uhol vektora napätia (53.)

Obrázok 218, Transformácia napätí do priestorového vektora SSV (Single Space Vector)



Obrázok 219, Grafické znázornenie aproximácie – náhrady vektora $\underline{u}_s$ pomocou $\underline{u}_1$ a $\underline{u}_2$.

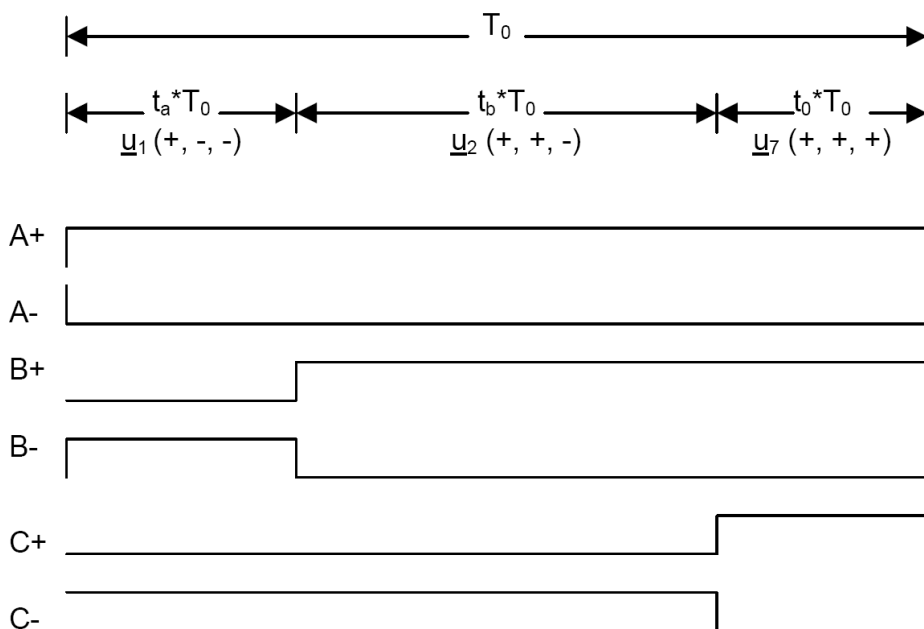


Ak modulačný index je malý, tak $(t_a + t_b) < 1$ a následne vzťah $t_a \cdot \underline{u}_1 + t_b \cdot \underline{u}_2$ je menší ako T_0 .

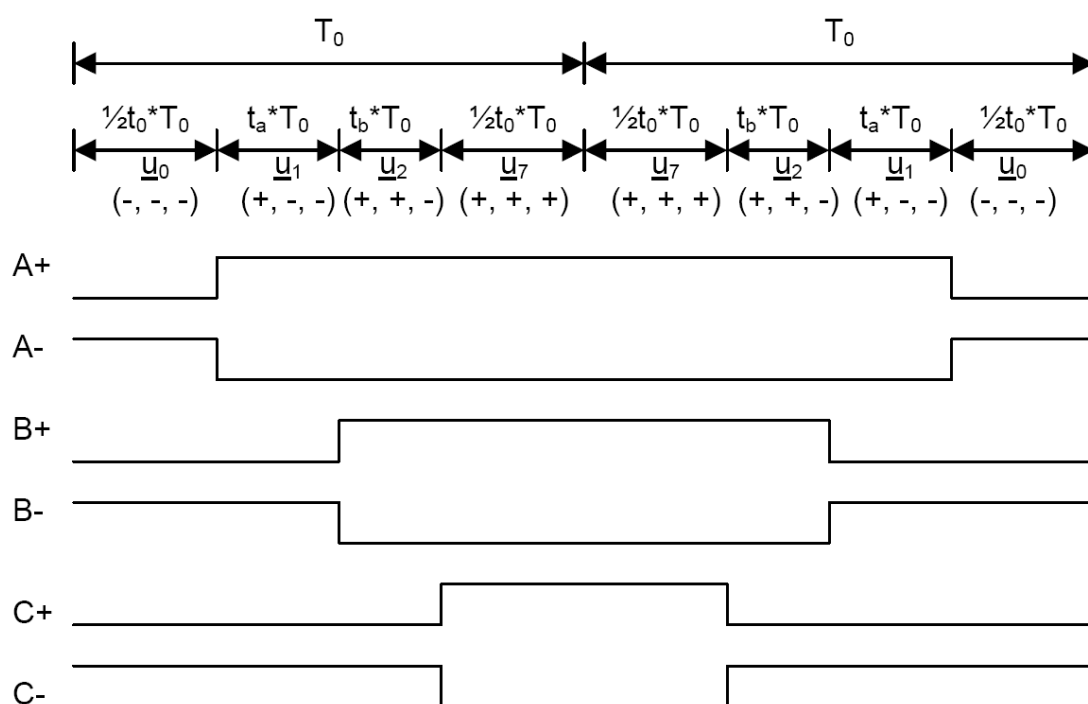
Potom môžeme vyjadriť t_0 ako

$$t_0 = T_0 \cdot (1 - t_a - t_b) \quad (54.)$$

Obrázok 220, Generovanie napätia pomocou metódy aproximácie vektora $\underline{u}_s$.



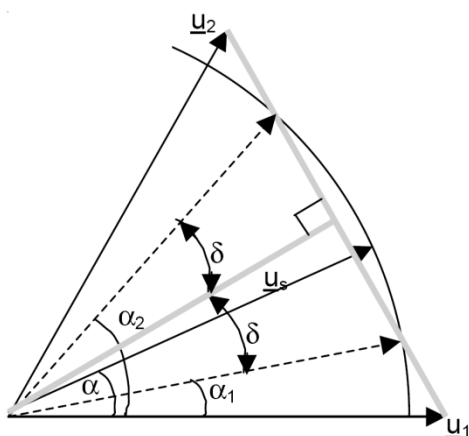
Obrázok 221, Generovanie napätia s modif. SSVM (Symetric Space Vector Modulation).



17.10.6. Zvyšovanie napätia SVM

Overmodulation je moderná technika, ktorá umožňuje SVM zvýšiť vyššiu amplitúdu napätia oproti SWPWM. Modulačný index U je obmedzený na hodnotu ~ 0.866 . Táto technika je používaná na rozšírenie rozsahu modulačného indexu. Overmodulation nastane, ak veľkosť vektora (šípka vektora) $\underline{u}_s$ sa nachádza mimo kružnice vpísanej do šesťuholníka. Obrázok 222 predstavuje jednoduchú metódu založenú na uhle α referenčného vektora $\underline{u}_s$. Ak uhol $\alpha < \alpha_1$ tak SVM je vykonávané s obvyklým U . Ak sa vektor $\underline{u}_s$ nachádza vo vnútri šesťuholníka t_0 je väčšie ako nula. Ak ale α sa nachádza medzi α_1 a $\frac{\pi}{6}$, hodnota t_0 môže byť menšie ako nula. K správnej činnosti je potrebné použiť pri SVM nový uhol α_1 a U . Ak ale α sa nachádza medzi $\frac{\pi}{6}$ a α_2 je potrebné použiť nový uhol α_2 a U , pretože t_0 je menšie ako nula. Použitie tohto spôsobu riadenia SVM sa zvýši veľkosť množstva harmonických zložiek a skreslenie výstupného napätia na výstupných svorkách výkonového meniča. Výstupné napätie sa týmto zvýši o hodnotu približne 12% oproti SVM bez overmodulation.

Obrázok 222, Grafické znázornenie priebehu vektora s modulačným indexom $U > 0.866$



$$\alpha_1 = \frac{\pi}{6} - \delta \quad (55.)$$

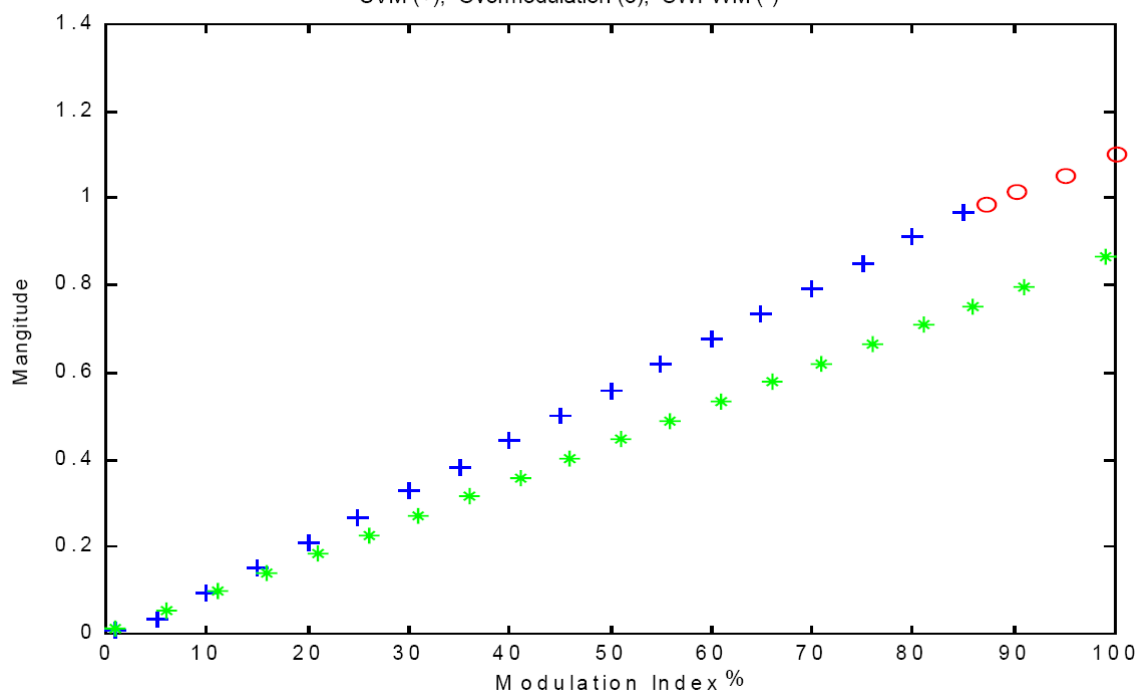
$$\alpha_1 = \frac{\pi}{6} + \delta \quad (56.)$$

$$\delta = \arccos\left(\frac{\sqrt{3}}{2 \cdot U}\right) \quad (57.)$$

Angle of $\underline{u}_s$ (α)	Modulation Index	Angle
$\alpha_1 < \alpha < \alpha_2$	U	α
$\alpha_1 < \alpha < \pi/6$	U	α_1
$\pi/6 < \alpha < \alpha_2$	U	α_2

Obrázok 223, Veľkosti napätí pri použití metód SWPWM, SVM a Overmodulation

SVM (+), Overmodulation (o), SWPWM (*)



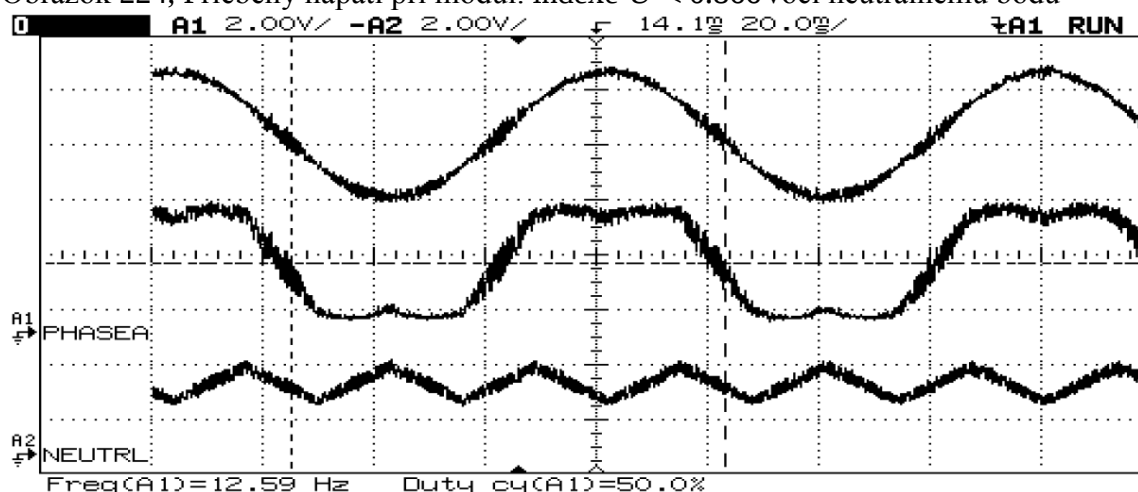
Tabuľka 16, Zoznam funkcií a použitých premenných pre SVM

Variable	Digital Representation	Engineering Unit Resolution	Comments
U ($ u_s $)	8-bit value (0x00 – 0xFF)	1/255 (~0.00392)	Input Variable
α ($\angle u_s$)	11-bit value (0x0000 – 0x05FF)	$\pi/765$ radians (~0.235 degrees)	Bits 0-7 contain α , Bits 8 – 10 contain sector (input variable)
T₀	8-bit value (0x00 – 0xFF)	CCU timer resolution (100 ns)	~19.6 kHz Carrier Frequency (fixed)
t_b	8-bit value (0x00 – 0xFF)	CCU timer resolution (100 ns)	Stored in a table Table contains t _b for U = 1.0 (0xFF)
t_a	8-bit value (0x00 – 0xFF)	CCU timer resolution (100 ns)	Stored in a table t _b table is used in reverse
t₀	8-bit value (0x00 – 0xFF)	CCU timer resolution (100 ns)	Calculated $t_0 = \sim(U*t_a + U*t_b)/256$
δ	7-bit value (0x00 – 0x7F)	$\pi/765$ radians (~0.235 degrees)	Stored in a table
α_1 and α_2	8-bit values (0x00 – 0xFF)	$\pi/765$ radians (~0.235 degrees)	Calculated as needed from δ

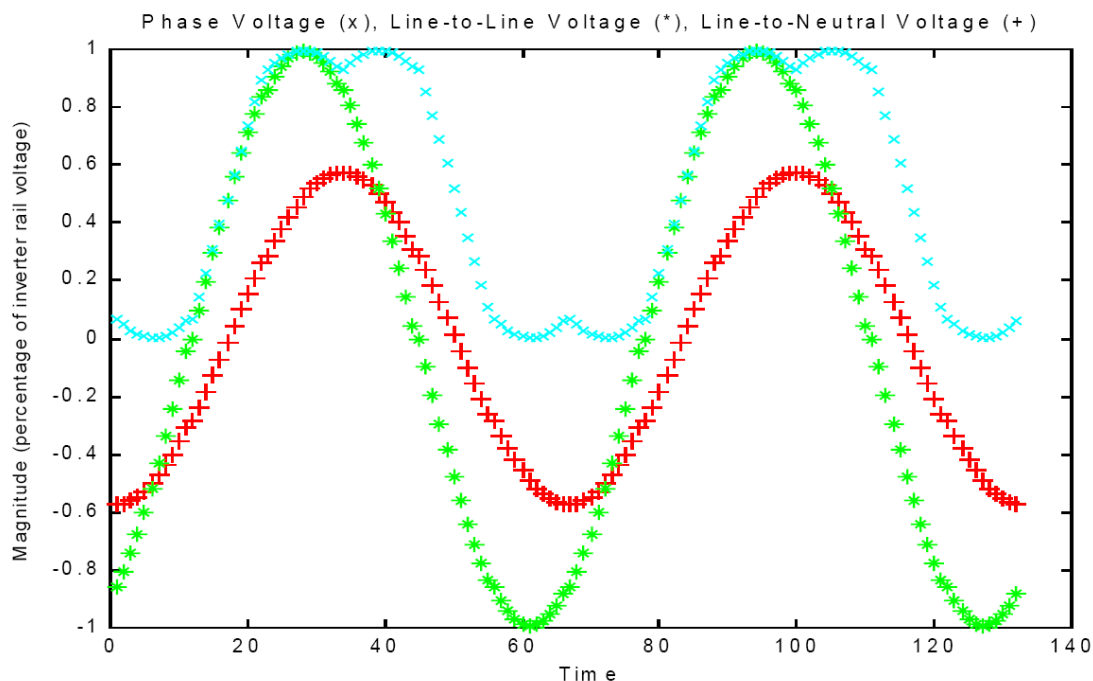
Tabuľka 17, Zoznam funkcií a premenných pre jednotlivé kvadranty SVM

Sector	Phase A Compare Value	Phase B Compare Value	Phase C Compare Value
0	$\frac{1}{2} t_0$	$\frac{1}{2} t_0 + t_a$	$T_0 - \frac{1}{2} t_0$
1	$\frac{1}{2} t_0 + t_b$	$\frac{1}{2} t_0$	$T_0 - \frac{1}{2} t_0$
2	$T_0 - \frac{1}{2} t_0$	$\frac{1}{2} t_0$	$\frac{1}{2} t_0 + t_a$
3	$T_0 - \frac{1}{2} t_0$	$\frac{1}{2} t_0 + t_b$	$\frac{1}{2} t_0$
4	$\frac{1}{2} t_0 + t_a$	$T_0 - \frac{1}{2} t_0$	$\frac{1}{2} t_0$
5	$\frac{1}{2} t_0$	$T_0 - \frac{1}{2} t_0$	$\frac{1}{2} t_0 + t_a$

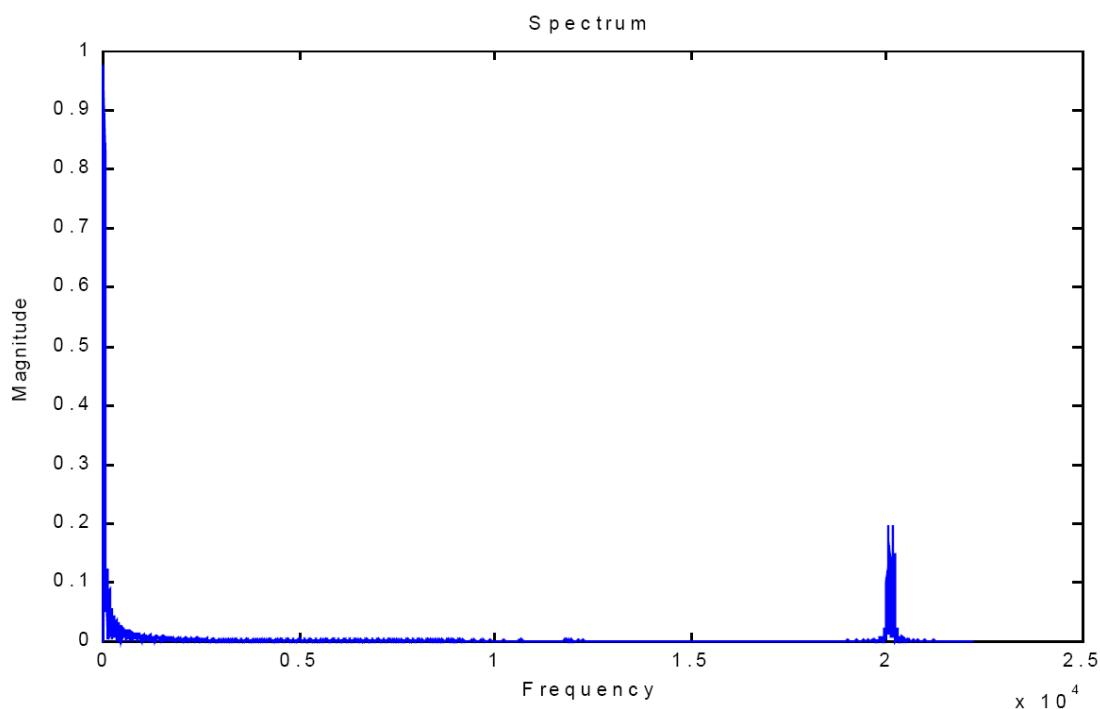
Obrázok 224, Priebehy napätí pri modul. indexe $U < 0.866$ voči neutrálnemu bodu



Obrázok 225, Priebeh napätia modulovaného pomocou SVM (Space Vector Modulation)



Obrázok 226, Obsah vyšších harmonických zložiek s SVM pri modul. indexe $U \sim 0.866$



Poznámka: bližšie informácie¹⁴⁷ je možné získať na www.infineon.com.

147

http://www.infineon.com/dgdl/ap0803620_Space_Vector_Modulation.pdf?folderId=db3a304412b407950112b40c497b0af6&fileId=db3a304412b407950112b4199b7b28b3&sId=db3a30433b47825b013b5c7695e449ee

17.11. Zdieľanie sériového kanála viacerými úlohami

Príklad: Napíšte program, ktorý bude zdieľať sériové rozhranie RS232 súčasne viacerými úlohami a v periodických intervaloch bude vysielat' údaje o momentálne spustenej úlohe na sériovú linku.

Príklad programu:

C:\Keil\C51\Examples\termel\Serials\PfSema\main.c

```
1  #include <rtx51.h>
2  #include <reg51.h>
3  #include <stdio.h>
4
5  // Task Definitions
6  #define STARTUP_TASK    3
7  #define TASK1           1
8  #define TASK2           2
9
10 // Semaphore Definitions
11 #define SEM_PRINTF      8
12
13 #define USE_SEMAPHORE   1
14
15 /**
16 make sure to link with...
17 ol(* ! ?PR?PRINTF?PRINTF)
18 ***/
19
20 void setup_serial_io (void)
21 {
22     SCON = 0x50;          /* SCON: mode 1, 8-bit UART, enable rcvr */
23     TMOD |= 0x20;         /* TMOD: timer 1, mode 2, 8-bit reload */
24     TH1 = 0xF0;          /* TH1: reload value */
25     TR1 = 1;             /* TR1: timer 1 run */
26     TI = 1;              /* TI: set TI to send first char of UART */
27 }
28
29 void task_1 (void) _task_ TASK1 _priority_ 0
30 {
31     while (1)
32     {
33         os_wait (K_IVL, 150, 0);          // Delay for 250 ticks
34
35         #if USE_SEMAPHORE
36             os_wait (K_MBX + SEM_PRINTF, 255, 0);          // Wait for printf
37             printf ("Task # %d\n", (int) os_running_task_id ());
38             os_send_token (SEM_PRINTF);          // Done with printf
39         #else
40             printf ("Task # %d\n", (int) os_running_task_id ());
41         #endif
42     }
43 }
44
```

C:\Keil\C51\Examples\termel\Serials\PfSema\main.c

```
45 void task_2 (void) _task_ TASK2 _priority_ 1
46 {
47     while (1)
48     {
49         os_wait (K_IVL, 100, 0); // Delay for 200 ticks
50
51     #if USE_SEMAPHORE
52         os_wait (K_MBX + SEM_PRINTF, 255, 0); // Wait for printf
53         printf ("Task # %d\n", (int) os_running_task_id ());
54         os_send_token (SEM_PRINTF); // Done with printf
55     #else
56         printf ("Task # %d\n", (int) os_running_task_id ());
57     #endif
58     }
59 }
60
61 void startup_task (void) _task_ STARTUP_TASK _priority_ 2
62 {
63     os_set_slice (1000);
64
65     os_create_task (TASK1);
66     os_create_task (TASK2);
67
68     #if USE_SEMAPHORE
69         os_send_token (SEM_PRINTF); // Setup printf semaphore
70     #endif
71
72     os_delete_task (os_running_task_id ());
73
74     while (1)
75     {
76         /** This should never happen ***/
77     }
78 }
79
80 void main (void)
81 {
82     setup_serial_io ();
83     printf ("Getting ready to start\n\n");
84
85     os_start_system (STARTUP_TASK);
86
87     while (1)
88     {
89         /** This should never happen ***/
90     }
91 }
```


Príklad programu pre obsluhu AD prevodníka v jazyku C:

C:\Omega\data\Dropbox\Záloha\C51\termel\Usmernovac\Usmernovac.c

```
46  int ADC (unsigned char Channel )
47  {
48      #define ADCS  0x08
49      #define ADCI  0x10
50      #define ADEX  0x20
51      ADCON =Channel ;           //Vyber kanalu pre prevod 0..7
52      ADCON |=ADCS ;             //Spustim prevod, bolo tu ADCON|=ADEX+ADCS;
53      while ((ADCON &ADCI) == 0x00 );    //Cakam na ukoncenie prevodu cca. 80us
54      ADCON ^=ADCI ;             //Nulujem priznak AD prevodnika
55      return (256 *ADCH +( ADCON &0xC0 )>> 6);
56  }
```

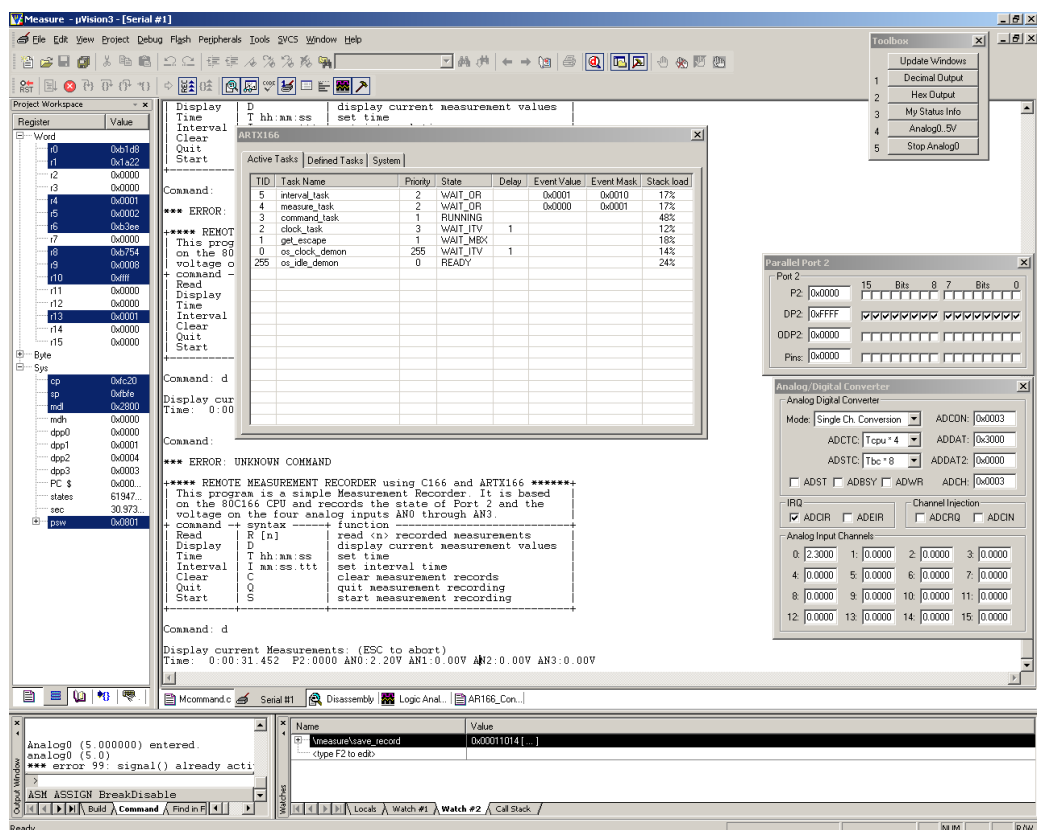
Poznámka: program je napísaný ako funkcia vracajúca **int**, kde parametrom je číslo kanálu pre prevod analógovej veličiny.

Príklad programu pre obsluhu AD prevodníka pomocou RTX51:

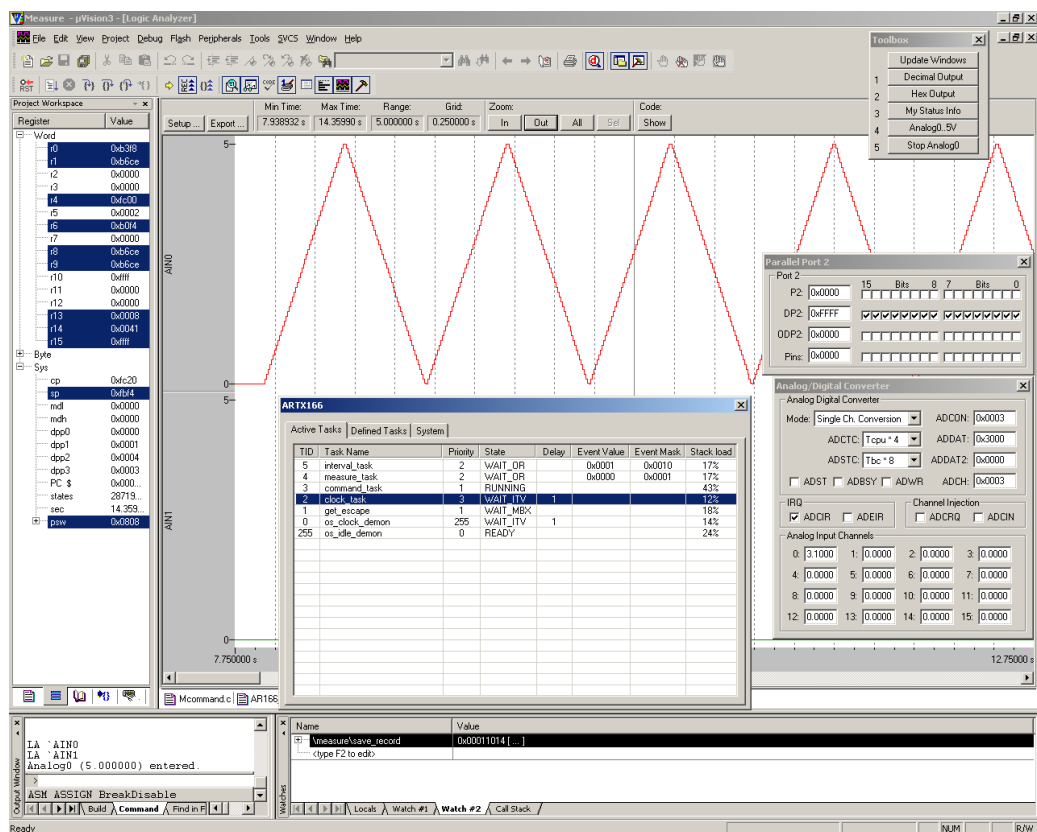
C:\Omega\data\Dropbox\Záloha\C51\i2c_dsw\i2c_DSW.c

```
46  #define ADCI  0x10
47  #define ADCS  0x08
48
49  unsigned int Voltage ;
50
51  #ifdef ADCINT
52
53  #pragma REGISTERBANK (3)
54
55  void Adc (void) _task_ ADC _priority_ 3
56  {
57      signed char RtxCompletion ;
58      os_attach_interrupt (10);
59      while (1)
60      {
61          Watchdog ;
62          switch (RtxCompletion =os_wait (K_TMO+K_INT+K_SIG ,10 ,NULL ))
63          {
64              case SIG_EVENT : //Voltage=ADC_Conversion(2); break;
65              case TMO_EVENT : ADCON |=( ADCS +2);
66              case INT_EVENT  : ADCON ^=ADCI ; Voltage =Conversion (); break ;
67          }
68      }
69  }
70
71  #pragma REGISTERBANK (0)
72
73  #else
74
75  #pragma REGISTERBANK (0)
76
77  void Adc (void) _task_ ADC _priority_ 0
78  {
79      signed char RtxCompletion ;
80      while (1)
81      {
82          Watchdog ;
83          Voltage =ADC_Conversion (2);
84          RtxCompletion =os_wait (K_TMO ,10 ,NULL );
85      }
86  }
87
88  #endif
```

Obrázok 227, Ukážka grafického výstupu programu na meranie jednosmerného napätia



Obrázok 228, Ukážka priebehu vstupného napätia pre AD vodičnik mikroprocesora



17.13. Harmonická analýza analógových veličín s C51 a RTX51

Rozbor príkladu: Harmonickou analýzou analógového priebehu je možné zistiť veľkosti amplitúd jednotlivých harmonických napätí obsiahnutých v neharmonickom napäťovom priebehu. Vytvorte program, ktorý bude v pravidelných periodických intervaloch $\Delta T=1,0$ ms. vzorkovať napätie na vstupe analógového prevodníka P5.0 mikroprocesora 80C552, zároveň bude vypočítavať Fourier-ove¹⁴⁸ koeficienty daného analógového priebehu a zobrazovať ich cez sériové rozhranie RS232 na LCD zobrazovači.

Pre neharmonický prúd zátáže môžeme napísať :

$$i_{ZAT} = \frac{a_0}{2} + \sum_{n=1}^{\infty} (a_n \cdot \cos(\omega_n \cdot t)) + \sum_{n=1}^{\infty} (b_n \cdot \sin(\omega_n \cdot t)) \quad (58.)$$

kde a_n a b_n sú koeficienty Fourier-ovho radu a a_0 je jednosmerná zložka analyzovaného priebehu. Amplitúda a fázový posun prvej harmonickej zátážového prúdu sú:

$$I_{1ZAT} = \sqrt{a_1^2 + b_1^2}; \varphi_1 = \arctan\left(\frac{b_1}{a_1}\right) \quad (59.)$$

Amplitúda sieťového prúdu s nulovým fázovým posunom oproti napájaciemu napätiu je daná ako:

$$I_{1SIE} = I_{1ZAT} \cdot \cos \varphi_1 \quad (60.)$$

Pre určenie prvej harmonickej zátážového prúdu a jeho fázového posunu využijeme Park-Clarke-ovú transformáciu. Potom:

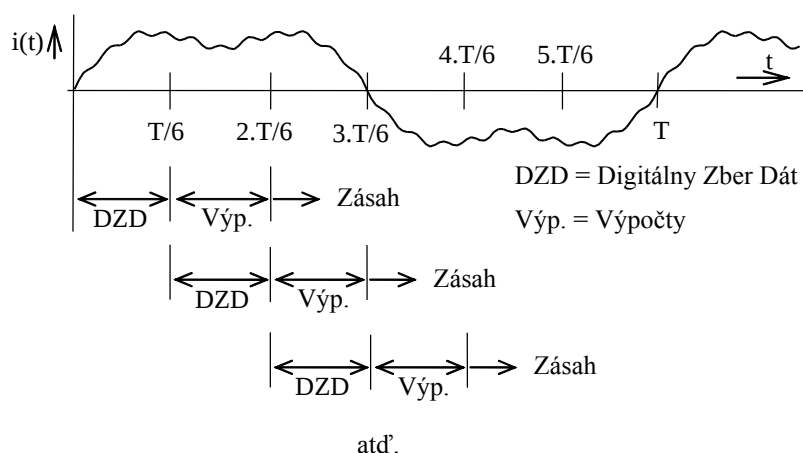
$$i_{ZAT}(t) = i_{\alpha} + j \cdot i_{\beta} \quad (61.)$$

Dá sa dokázať, že pre **symetrickú trojfázovú zátáž** je možné súradniciach α, β vypočítať komplexné koeficienty Fourier-ovho radu vzoriek získaných v $\frac{1}{6}$ periódy zátážového prúdu.

Tým sa výrazne zvýši dynamika celého aktívneho filtra, lebo ak výpočet koeficienta pre prvú harmonickú prúdu zátáže (C_1) stihne riadiaci systém za ďalšiu šestinú, tak v nasledujúcej šestine môže byť vykonaný zásah do sústavy. Nasledujúci obrázok ukazuje zretáženie jednotlivých celkov riadiaceho algoritmu.

¹⁴⁸ <http://matematika.cuni.cz/dl/analyza/37-fou/lekce37-fou-pmax.pdf>

Obrázok 229, Sekvencie riadiaceho algoritmu aktívneho filtra



Pre výpočet komplexných koeficientov Fourier-oveho radu platí nasledujúci vzťah:

$$C_v = \frac{3}{\pi} \int_0^{\pi/3} RE\{f(\omega t)\} \cdot e^{-jv\omega t} \cdot d\omega t + j \frac{3}{\pi} \int_0^{\pi/3} IM\{f(\omega t)\} \cdot e^{-jv\omega t} \cdot d\omega t \quad (62.)$$

kde $RE = i_\alpha$ a $IM = i_\beta$ ďalej:

$$C_1 = \frac{3}{\pi} \int_0^{\pi/3} [(RE \cdot \cos(\omega t) + IM \cdot \sin(\omega t)) + j \cdot (IM \cdot \cos(\omega t) - RE \cdot \sin(\omega t))] d\omega t \quad (63.)$$

a po prechode na diskretný tvar dostaneme:

$$\begin{aligned} C_1 = & \frac{1}{N} \left\{ \sum_{i=0}^{N-1} RE(i) \cdot \cos\left(\frac{i \cdot \pi}{3 \cdot N}\right) + \sum_{i=0}^{N-1} IM(i) \cdot \sin\left(\frac{i \cdot \pi}{3 \cdot N}\right) + \right. \\ & + \frac{RE(N) \cdot \cos(\pi/3) - RE(0) \cdot \cos(0)}{2} + \\ & + \left. \frac{IM(N) \cdot \sin(\pi/3) - IM(0) \cdot \sin(0)}{2} \right\} + \\ & + j \frac{1}{N} \left\{ \sum_{i=0}^{N-1} IM(i) \cdot \cos\left(\frac{i \cdot \pi}{3 \cdot N}\right) - \sum_{i=0}^{N-1} RE(i) \cdot \sin\left(\frac{i \cdot \pi}{3 \cdot N}\right) + \right. \\ & + \frac{IM(N) \cdot \cos(\pi/3) - IM(0) \cdot \cos(0)}{2} - \\ & - \left. \frac{RE(N) \cdot \sin(\pi/3) + RE(0) \cdot \sin(0)}{2} \right\} \end{aligned} \quad (64.)$$

Túto rovnicu je nutné vyriešiť pomocou FFT¹⁴⁹ v reálnom čase, pričom je jasné, že čím viac vzoriek v danom intervale nasnímame, tým dlhšie bude trvať výpočet¹⁵⁰. Pre konkretizáciu ešte uvedieme, že výsledkom je komplexné číslo tvaru:

$$C_1 = a_1 + j \cdot b_1 \quad (65.)$$

¹⁴⁹ FFT (Fast Fourier Transform) je implementovaná v najrozšírenejších matematických programoch ak sú napríklad Matlab, MathCad, Mathematica a iné.

¹⁵⁰ Pre výpočty je vhodné použiť procesory s integrovanými numerickými procesormi, ktoré umožnia rapídne zvýšiť rýchlosť výpočtu zložitých matematických funkcií.

a dosadením do vzťahov¹⁵¹ dostaneme hodnotu prvej harmonickej sieťového prúdu. Ďalej analogicky dostávame jednotlivé hodnoty referenčného prúdu pre aktívny filter.

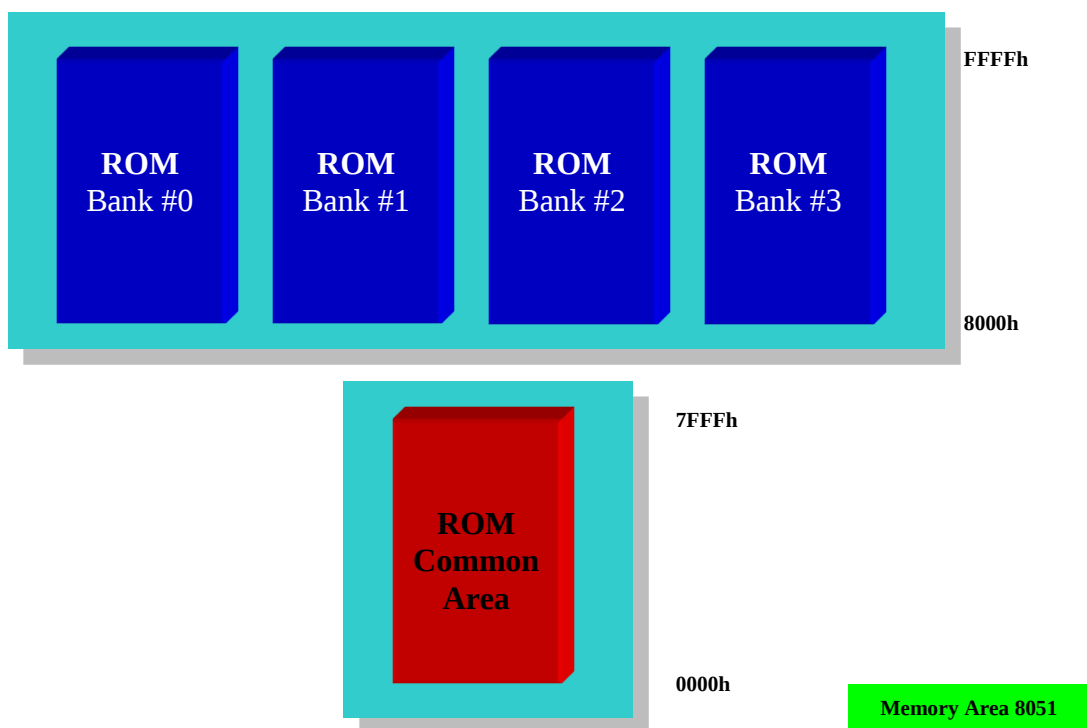
¹⁵¹ Rýchlosť výpočtu matematických vzťahov v tomto prípade má absolútnu prioritu, preto je na uvážení programátora či použije pomerne neefektívnu architektúru mikroprocesora 8051 bez operačného systému RTX51 a nesiahne radšej po modernejšej 32 bitovej ARM architektúre s podporou operačného systému RLARM verzie minimálne 4.12 alebo vyššej.

17.14. Práca s externým pamäťovým priestorom xBANKING

Príklad: Napíšte program ktorý bude pracovať so sériovým kanálom a ktorého funkcie budú uložené v dvoch fyzicky oddelených externých pamätiach programu nazývaných BANK.

Rozbor príkladu: Tvorba programov s použitím technológie **xBANKING** pri jedno čipových mikroprocesoroch 8051 umožňuje používať externé pamäťové priestory presahujúce maximálnu hranicu pre 8051 t.j. 64kB programu a dát. K adresácii tohto pamäťového priestoru sa používajú prídavné adresné vodiče portu P1. V závislosti na type procesora je potom možné pracovať až s 256 bankami externej pamäti programu a dát.

Obrázok 230, Pamäťový model externej pamäti programu - xBANKING



Vyššie uvedený príklad ukazuje možnosť pripojiť k mikroprocesoru 8051 navyše ešte 4 banky pamäti¹⁵² programu ROM a 4 banky pamäti dát RAM. Pomocou linkera LX51 je možné vygenerovať výsledný program pre mikroprocesor 8051 vo formáte INTEL HEX, alebo HEX-386, ktorý môžeme zapísať do pamäti s veľkosťou presahujúcou 64kB. Týmto je možné vytvoriť pamäť pozostávajúcu z niekoľkých fyzických segmentov do jednej. Obrázok 230 to názorne zobrazuje.

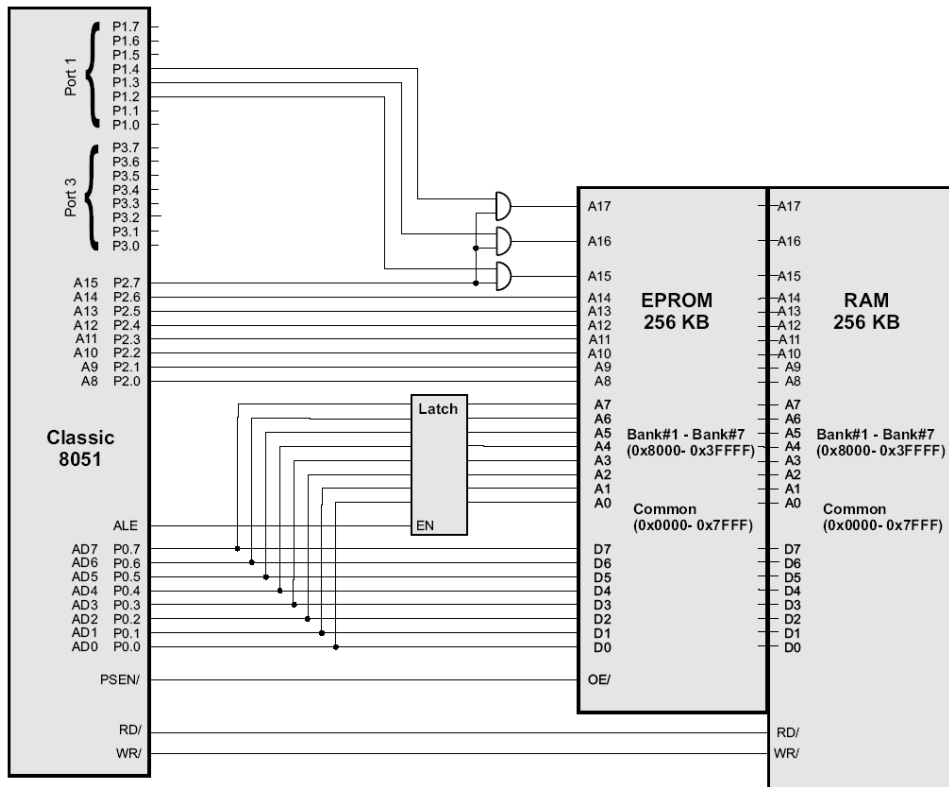
¹⁵² Mikroprocesory DS80C390, DS80C400, DS80C410 a DS80C411 majú možnosť lineárneho adresovania oproti štandardnej 16 bitovej adresy 8051 pomocou úplnej 24 bitovej adresy t.j. až 16MB pamäti ROM, alebo RAM.

Obmedzenia použitia pre Common Code Area Bank (CCAB):

- Inicializácia po RESET-e a adresy vektorov prerušení musia byť umiestnené v CCAB, pretože na rozdiel od mikroprocesora 8051 linker LX51 nedokáže vopred zistiť či prišiel signál RESET, alebo prerušenie. Z tohto dôvodu linker LX51 umiestňuje vektory prerušenia a inicializačnú sekvenciu do CCAB.
- Konštanty programu (reťazce, znaky, číselné hodnoty, tabuľky ...) musia byť umiestnené v CCAB, pretože musí mať k nim 8051 prístup z ľubovoľnej časti programu, alebo banky programu. Dodatočne je možné konštanty relokovat' z CCAB do jednotlivých bánk, ale môže dochádzať k nekonzistentnosti údajov navzájom.
- Prerušená musia byť umiestnené vždy v CCAB, pretože prerušenia môžu byť vyvolané z ľubovoľnej časti programu, banky. Kompilátor C51 hlási varovanie ak obslužná rutina prerušenia je deklarovaná v niektorej banke programu.
- Bank Switch Code – obslužný program riadenia xBANKING je kompilátorom umiestňovaný v CCAB a prepínanie je uskutočňované pomocou špeciálnej tabuľky (Switching Code Table) skokov ku ktorej majú prístup všetky banky. Nie je možné pristupovať do tejto tabuľky z jednotlivých bánk. Prístup do tabuľky je umožnený len z CCAB.
- Library Functions – knižnice podprogramov, funkcií použité kompilátorom Cx51, alebo PL/M-51 musí byť umiestnené v CCAB. Pri odovzdávaní hodnôt medzi CCAB a niektorou bankou, alebo vzájomnom prenose návratových hodnôt funkcií sú používané registre mikroprocesora 8051. Linker Lx51 umiestňuje tento kód len do CCAB a nenachádza sa v žiadnej banke! Z tohto dôvodu je veľmi obtiažné určiť veľkosť výsledného programového kódu, pretože jeho veľkosť závisí na použitom hardware 8051 a množstva použitých funkcií. Linker Lx51 niekedy nedokáže účinne odstrániť duplicitné časti programu z niektorých bánk, čo v konečnom dôsledku môže spôsobiť nedostatok systémových prostriedkov pamäte mikroprocesora 8051.
- Prepnutie medzi bankami vyžaduje približne 50 cyklov 8051 a spotrebuje 2 byte na vrchole zásobníka. Z tohto dôvodu sa odporúča, aby často vykonávané procedúry a funkcie boli umiestňované v CCAB a nie v niektorej banke.

Bližšie informácie o výhodách a nevýhodách xBANKINGu je možné nájsť na www.keil.com.

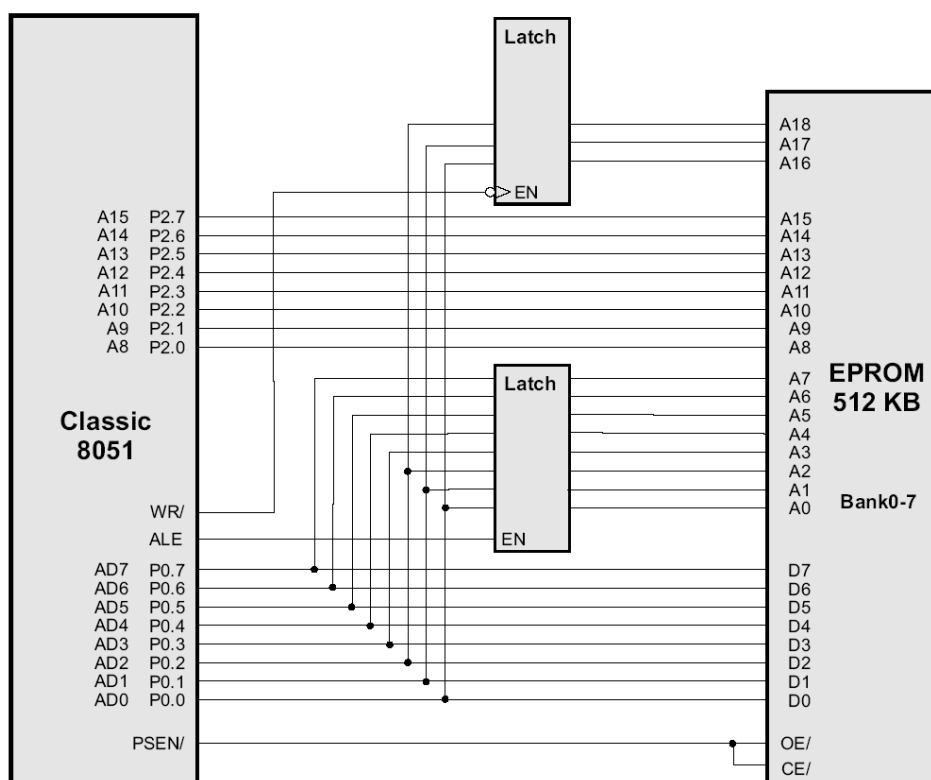
Obrázok 231, Zapojenie pamäti 256kB RAM a EPROM k 8051, 8x32kB.



Vyššie uvedený obrázok ukazuje možnosť využitia prídavnej pamäti RAM a ROM celkovo s veľkosťou 256kB. Adresovanie je vytvorené pomocou prídavných adresných vodičov A16, A17, A18 ktoré vznikli logickým súčinom A15 a príslušnými P1.2, P1.3, P1.4.

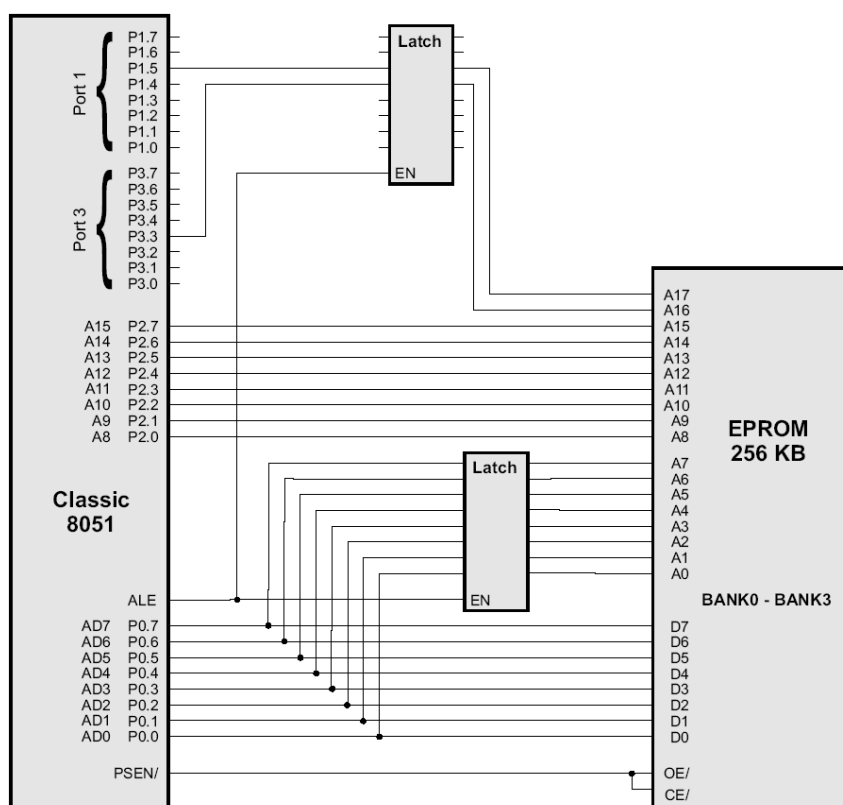
Tento spôsob má oproti iným zapojeniam veľkú výhodu, pretože okrem pamäti programu zväčšuje aj pamäť dát. Takto môže programátor mať k dispozícii pre každú banku programu aj vlastnú banku dát, ktorá je fyzicky oddelená. Určitou nevýhodou je, že maximálne množstvo pamäti ktoré môžeme týmto spôsobom využívať je 256kB v adresnom priestore 0000h až 7FFFh pre *Common Code Area Bank* a od 8000h do 3FFFFh pre program a dáta Bank0 až Bank7. Takže pre programátora je k dispozícii priestor v rozsahu 8000h až FFFFh t.j. 32kB pre Bank0 a 18000h až 1FFFFh pre Bank1 t.j. ďalších 32kB.

Obrázok 232, Zapojenie 512kB pamäti programu k 8051 pomocou adresovania XDATA



Tento spôsob adresovania umožňuje adresovať maximálne 512kB pamäti programu. Pre adresné vodiče A16 až A18 je použitý záchytný obvod LATCH, ktorý využíva ako strobovací (potvrdzovací) vodič zápisový signál $\overline{WR}$ pamäti dát mikroprocesora 8051. Signál ALE je použitý pre zachytenie 8 bitovej adresy ktorá je multiplexovaná na porte P0. Výberový signál $\overline{PSEN}$ je použitý pre pamäť programu. Programátor má k dispozícii pamäťový priestor pozostávajúci z ôsmich 64kB bánk programu v adresnom priestore 0000h až 80000h t.j. Bank0 až Bank7. Bank0 má k dispozícii pamäťový priestor v rozsahu adres 0000h až FFFFh, Bank1 10000h až 1FFFFh atď.

Obrázok 233, Zapojenie 256kB pamäti programu k 8051 pomocou štyroch 64kB bánk



Tento spôsob adresovania umožňuje adresovať maximálne 256kB pamäti programu. Signál *ALE* je použitý pre zachytenie 8 bitovej adresy ktorá je multiplexovaná na porte P0 a zároveň aj pre adresné vodiče A16 a A17 ktoré sú použité spolu s A15 pre vytvorenie 19 bitovej adresy pre rozšírený pamäťový priestor 8051. Výberový signál $\overline{PSEN}$ je použitý pre pamäť programu. Programátor má k dispozícii pamäťový priestor pozostávajúci zo štyroch 64kB bánk programu v adresnom priestore 0000h až 40000h t.j. Bank0 až Bank3. Pamäťový priestor pre Bank0 je v rozsahu 0000h až FFFFh, pre Bank1 10000h až 1FFFFh atď.

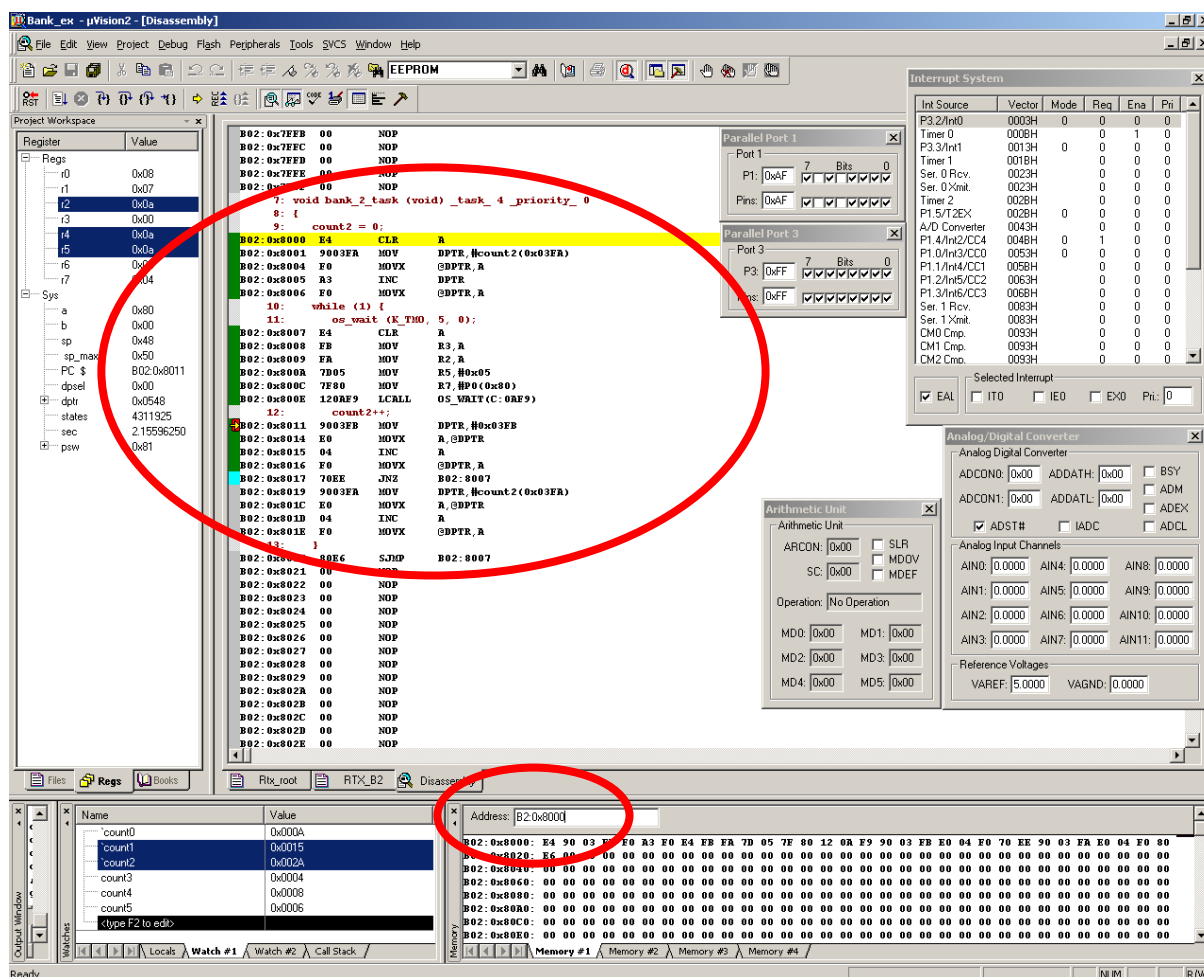
Obrázok 234, Popis premenných (Environments) pre xBANKING 8051

Name	Description
?B_NBANKS	number of banks to be supported. The following values are allowed: 2, 4, 8, 16, and 32. Two banks require one additional address (or I/O Port) line; four banks require two lines; eight banks require three lines; sixteen banks require four lines, and thirty-two banks require five address lines.
?B_MODE	indicates the way how the address extension is done. 0 for using an standard 8051 I/O Port, 1 for using an XDATA port; 2 for using 80C51MX address line; and 4 for user provide bank switch code.
?B_RTX	specifies if the application uses RTX-51 Full. Only ?B_MODE 0 and 1 are supported by RTX51 Full.
?B_VARBANKING	Enables variable banking in XDATA and CODE memory. Variable banking requires the LX51 linker/locater. It is not supported by BL51 . Refer to "Banking With Common Area" on page 303 for an example on how to setup the LX51 linker/locater.
?B_RST_BANK	Specifies the default bank that is selected after CPU reset. This setting is used by the LX51 linker/locater to reduce the entries in the INTERBANK CALL TABLE. The value 0xFF disables this optimization. This value is not used by BL51 .
For ?B_MODE = 0 (bank switching via 8051 I/O Port) define the following:	
?B_PORT	used to specify the address of the internal data port. The SFR address of an internal data port must be specified. (For example: P1 as for port 1).
?B_FIRSTBIT	indicates which bit of the 8051 I/O port is to be assigned first. The value 3 indicates that port bit 3 is used as first port line for the address extension. If, for example, two address lines are used, P1.3 and P1.4 are allocated in this case. The remaining lines of the 8051 I/O port can be used for other purposes.

Obrázok 235, Zoznam argumentov LX51 a BL51 pre xBANKING 8051

Argument	Description
address	A value representing a memory location. For BL51, L251 and LX51 in Philips 80C51MX mode plain numbers are used to represent an address. LX51 uses for classic 8051 devices a memory prefix in the address specification. For example: D:0x55 refers to DATA memory address 0x55 C:0x8000 refers to CODE memory address 0x8000 B4:0x4000 refers to CODE memory address 0x4000 in code bank 4.
classname	A name of a memory class. The x51 tools allows basic classes and user defined classes. Refer to "Memory Classes and Memory Layout" the page 27 for more information about memory classes.
filename	A file name that corresponds to the Windows file name conventions.
modname	A module name. Can be up to 40 characters long and must start with: A – Z , ? , _ , or @ ; following characters can be: 0 – 9 , A – Z , ? , _ , or @ .
range	An address range in the format: startaddress [– endaddress] The startaddress is the first address specified by the range. The endaddress is optional and specifies the last address which is included in the address range.
segname	A segment name. Can be up to 40 characters long and must start with: A – Z , ? , _ , or @ ; following characters can be: 0 – 9 , A – Z , ? , _ , or @ .
sfname	A segment or function name.
value	A number, for example, 1011B, 2048D, 0x1000, or 0D5FFh.

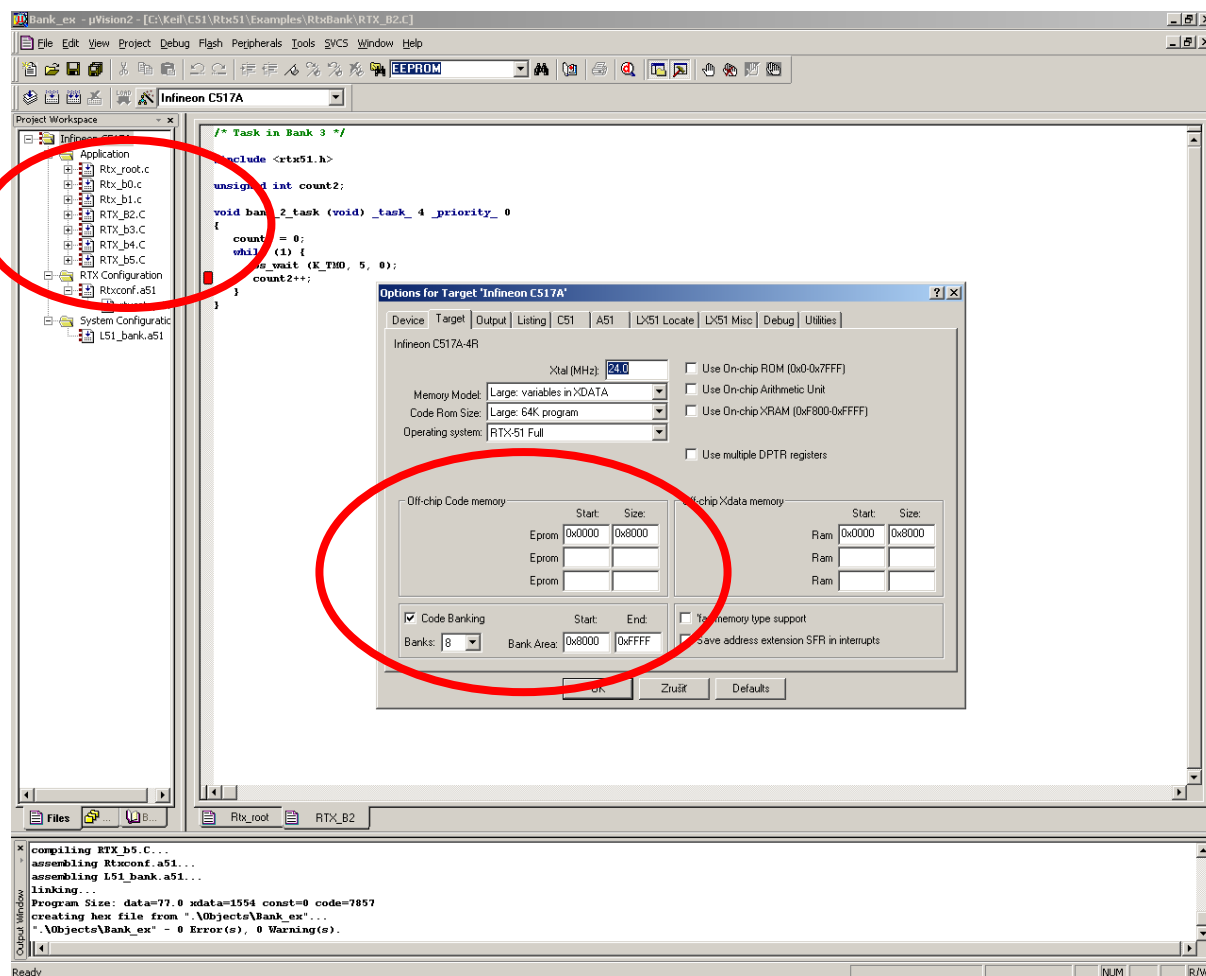
Obrázok 236, Ukážka odladovaného programu s RTX51 Full a xBANKING (6 bánk)



Vyššie uvedená ukážka zobrazuje princíp zdieľania a samotnej existencie pamäťového priestoru pre 8051. Principiálne tu existuje 6 bánk v ktorých je uložený program od adresy 8000h do FFFFh s periodickým offsetom 10000h. V pamäťovom priestore od 0000h až do 7FFFh s periodickým offsetom 10000h je uložený hlavný program obsahujúci hlavný kód programu a obsluhu pre xBANKING. Prepínanie pamäťového priestoru je vytvorené doplnením troch adresných vodičov A16 až A18, ktoré umožňujú adresovať pripojenú pamäť programu ROM, prípadne pamäť dát RAM.

Pri samotnom odladovaní programu v prostredí Keil μVision3 je potom adresa programu v rozšírenom pamäťovom priestore zobrazovaná v tvare B02:0x8000, kde prvá časť adresy nesie o informáciu o príslušnej banke programu a druhá časť jej adresu vo fyzickom pamäťovom priestore. Žltým pruhom je zobrazený začiatok programu uložený v príslušnej banke B02 na adrese 8000h.

Obrázok 237, Nastavenie prostredia KEIL μVision 3 pre prácu s xBANKING



Príklad programu C_ROOT.C:

```
/*-----  
C_ROOT.C  
  
Copyright 1995-1996 Keil Software, Inc.  
-----*/  
  
#include <stdio.h>  
#include <reg51.h>  
  
extern void func0(void);  
extern void func1(void);  
  
void main(void)  
{  
    /* INITIALIZE SERIAL INTERFACE TO 2400 BAUD @12MHz */  
    SCON = 0x52; /* SCON */  
    TMOD = 0x20; /* TMOD */  
    TCON = 0x69; /* TCON */  
    TH1 = 0xf3; /* TH1 */  
    printf("MAIN PROGRAM CALLS A FUNCTION IN BANK 0 \n");  
    func0();  
    printf("MAIN PROGRAM CALLS A FUNCTION IN BANK 1 \n");  
    func1();  
    while(1);  
}
```

Príklad programu C_BANK0.C:

```
/*-----  
C_BANK0.C  
Copyright 1995-1996 Keil Software, Inc.  
-----*/  
  
#include <stdio.h>  
  
extern void func2(void);  
  
void func0(void)  
{  
    printf("FUNCTION IN BANK 0 CALLS A FUNCTION IN BANK 2 \n");  
    func2();  
}
```

Príklad programu C_BANK1.C:

```
/*-----  
C_BANK0.C  
Copyright 1995-1996 Keil Software, Inc.  
-----*/  
  
#include <stdio.h>  
  
extern void func2(void);  
  
void func0(void)  
{  
    printf("FUNCTION IN BANK 0 CALLS A FUNCTION IN BANK 2 \n");  
    func2();  
}
```

Príklad programu C_BANK2.C:

```
/*-----  
C_BANK2.C  
Copyright 1995-1996 Keil Software, Inc.  
-----*/  
  
#include <stdio.h>  
  
void func2(void)  
{  
    printf("THIS IS A FUNCTION IN BANK 2! \n");  
}
```


18 Inštalácia programu µVision

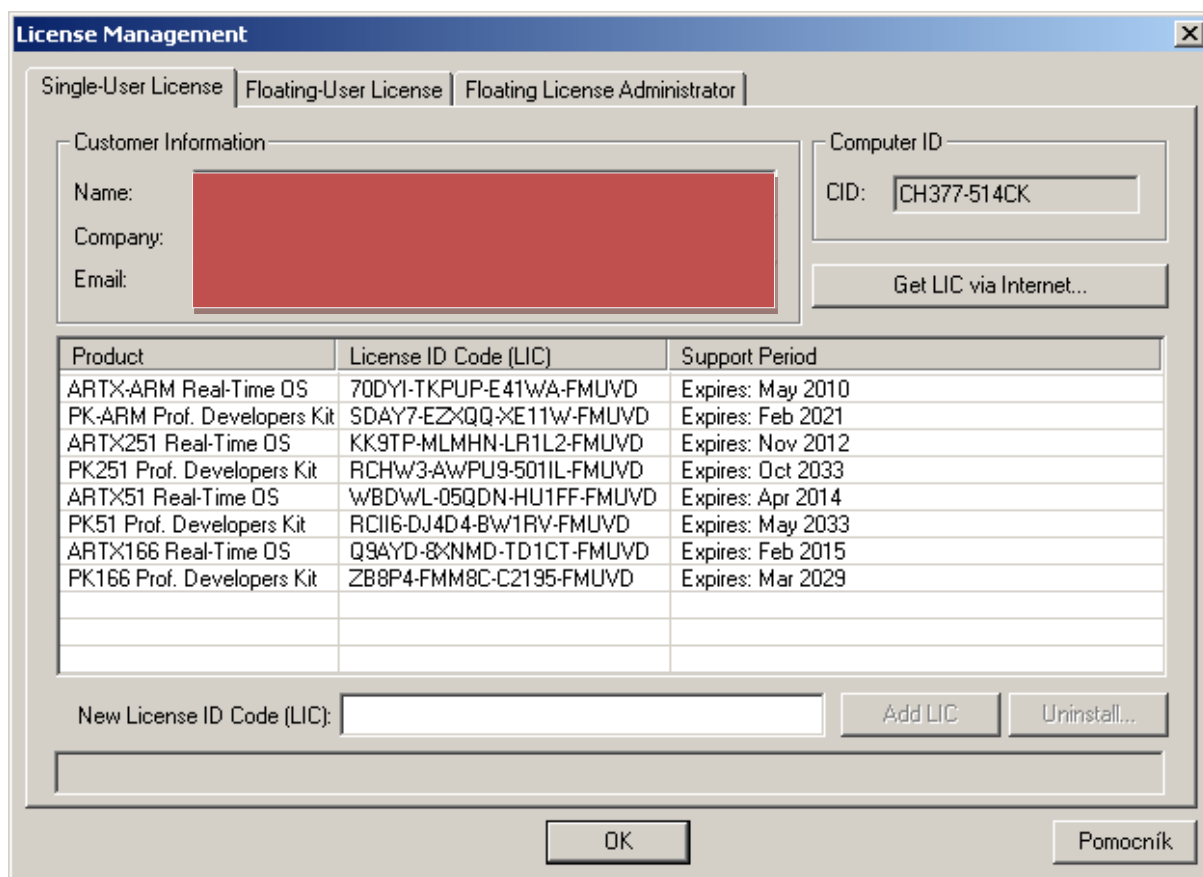
18.1. Inštalácia programu µVision2

Vyššie uvedený program je dodávaný štandardne na CDROM. Súčasťou dodávky je aj tzv. Addon disketa, ktorá obsahuje licenciu s unikátnym číslom, ktoré sa pri inštalácii odosiela pomocou internetu do ústredia firmy, ktorá inštalovaný produkt zaregistruje. Podľa typu licencie sa nainštaluje do PC zakúpený produkt. Práve táto disketa spolu s HW kľúčom „Dongle“ pripojenému k portu LPT určuje či sa bude inštalovať len A51, CA51, DK51, PK51. Inštalačné súbory nachádzajúce sa na CDROM sú pre všetky distribuované verzie. Celý inštalačný proces je automatizovaný, takže ho zvládne aj začiatočník. Ceny týchto produktov sa pohybujú od 375€ do 3500€. Ceny operačných systémov RTX od 350€ do 10000€. Ceny sú platné pre rok prvú polovicu roka 2005.

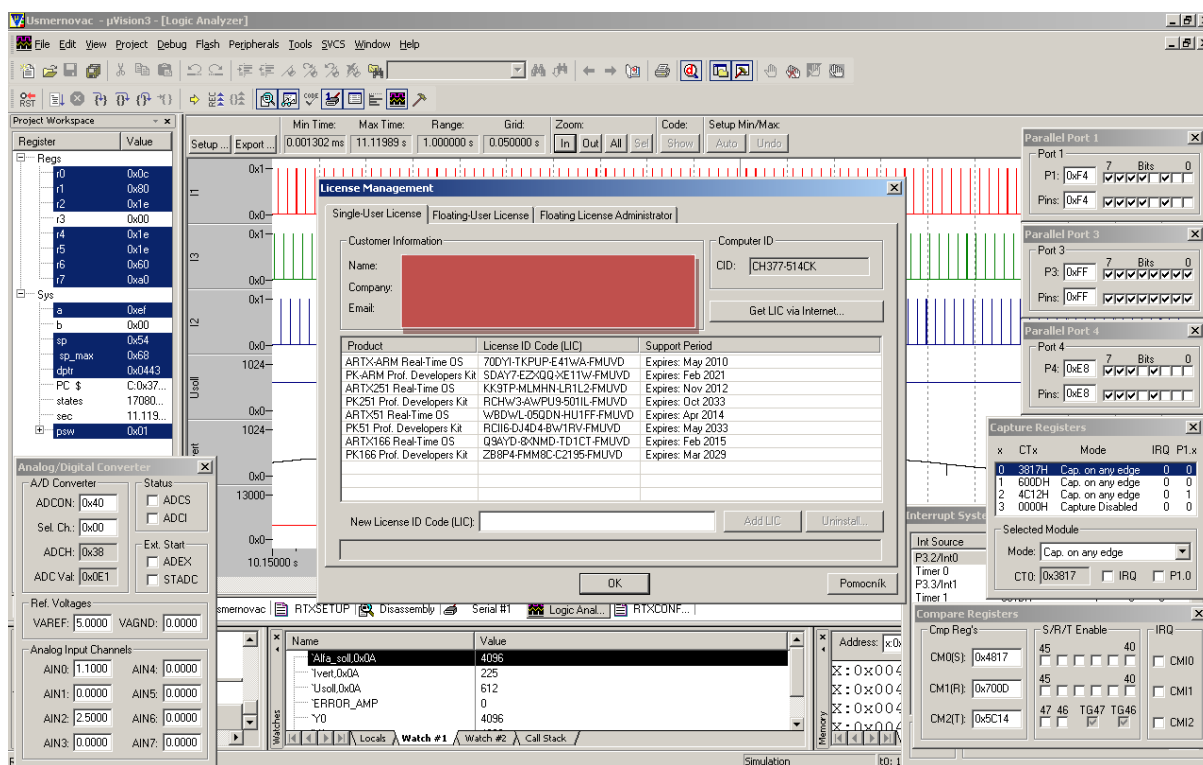
18.2. Inštalácia programu µVision3, µVision4 a vyšších verzií

Firma Keil Software inc. vzhľadom na prelomenie ochrany proti nelegálnemu kopírovaniu a neoprávnenému šíreniu svojich produktov pristúpila k zmene spôsobu nadobúdania licencií. K inštalácii produktov od firmy Keil je potrebné mať len inštalačné CDROM, HW kľúč a sériové číslo (Serial Number), ktoré je priložené v balíku priložené. Tieto produkty už nie je možné nainštalovať bez HW kľúča pripojeného k portu USB. Inštalácia je viazaná unikátnym číslom k danému PC. Toto číslo je generované podľa sériového čísla jednotlivých komponentov osadených v počítači. Pri inštalácii sa v menu License Management vygeneruje tzv. CID (Computer ID), ktoré sa odosiela do ústredia firmy Keil, ktorá po korektnom zaregistrovaní produktu odošle späť LIC (New License ID Code), ktoré sa zapíše do okna License Management. Od okamihu zadania LIC program poskytuje užívateľovi len tie programové produkty, ktoré má legálne zakúpené.

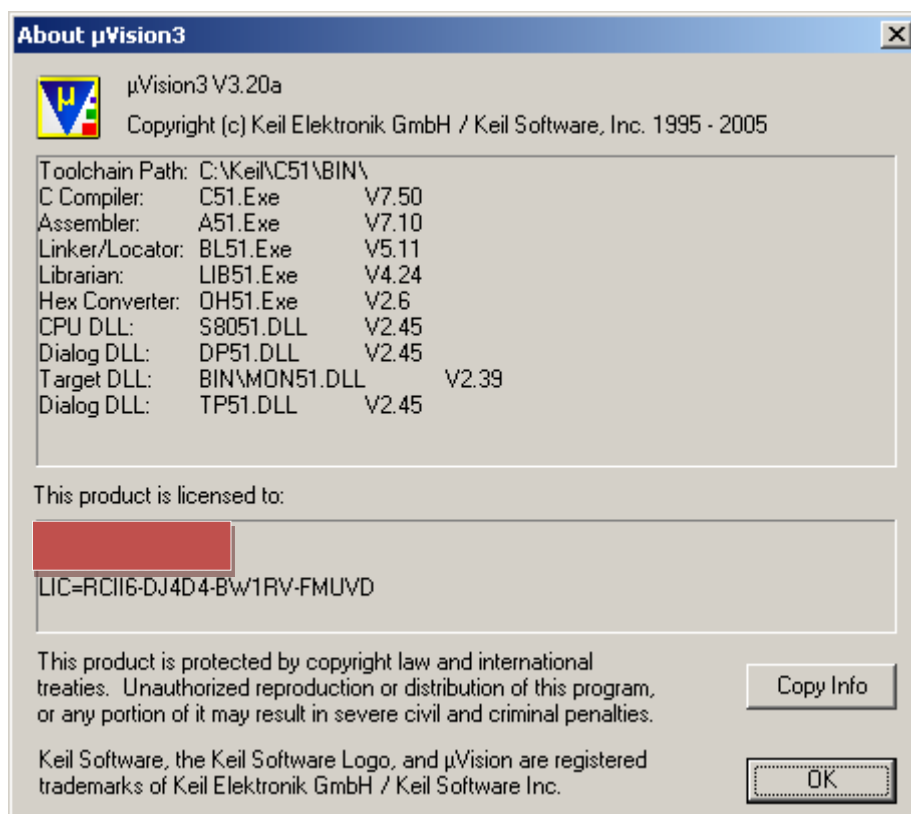
Obrázok 238, Správa a inštalácia licencií v prostredí µVision3



Obrázok 239, Zobrazenie licencií v prostredí µVision3



Obrázok 240, Informačné okno programu μ Vision3



Poznámka autora:

Práve v P2P sieťach bolo možné získať plne funkčnú „crack“ verziu vyššie uvedených produktov. Tie sú dostupné dodnes, ale už len v starších verziách neumožňujúcich pracovať v prostredí μ Vision3. Crack na μ Vision3 a μ Vision4 už existuje, ale vzhľadom na dobré a seriózne vzťahy s firmou Keil Software nie je ďalej distribuovaný. Tiež spôsob ochrany tohto produktu zatiaľ nebol doteraz zverejnený a podľa mojich vedomostí je tento „crack“ autorovým duševným vlastníctvom jedného programátora zo stredného Slovenska. Na prekonanie jednoduchšieho typu ochrany μ Vision2 potreboval cca. 9 mesiacov občasnej práce popri výkone vojenskej služby. Produkt ochrany produktu μ Vision3 už bol prekonaný za cca. 1 týždeň a μ Vision 4 doteraz nedoznal podstatnejších zmien.¹⁵³

Keďže vyššie uvedený programový produkt je naozaj výnimočným profesionálnym pomocníkom, budúcemu programátorovi isto nebude ľúto v budúcnosti vynaložiť finančné prostriedky na zakúpenie tohto nástroja.

¹⁵³ V súčasnosti je už k dispozícii prostredie μ Vision4 vo verzii PKC51 ver. 9.50a a vyššie uvedená ochrana je s malými obmenami opäť použitá. Od verzie 9.53a je už distribuované prostredie označované ako μ Vision5, ktoré je už oveľa prepracovanejšie a stabilnejšie ako predchádzajúce verzie.

19 Zoznam bibliografických odkazov

- Allworth, S.T.** *Introduction to Real-Time Software Design*. New York. : Springer-Verlag Inc.
- Boldea, I. a Nasar, S. A.** 1998. *Electric Drives*. s.l. : CD Interactive Version, 1998.
- Caha, Z. a Černý, M.** 1990. *Elektrické pohony*. Praha : SNTL, 1990.
- Deitel, H.M.** 1990. *Operating Systems*. 2. vydanie. s.l. : Addison-Wesley Publishing Company, 1990.
- Gieras, J. F.** 1994. „*Linear Induction Drives*“. New York : Oxford University Press Inc., 1994. ISBN 0-19-859381-3.
- Goldsmith, Sylvia.** *A practical guide to Real-Time Systems Development*. s.l. : Prentice Hall.
- Hrabovcová, V., a iní.** 2004. *Meranie a modelovanie elektrických strojov*. Žilina : EDIS - vydavateľstvo ŽU, 2004.
- Jadroň, Eduard.** 2015. *Príručka programovania v ASM a C s RTX51 od Diversant Software s príkladmi*. 2. Martin : Diversant Software, 2015. s. 260. DSW.
- Javůrek, J.** 2003. *Regulace moderních elektrických pohonů*. Praha : GRADA Publishing, 2003. ISBN 80-247-0507-9.
- Keil Elektronik GmbH, Mettler & Fuchs AG a Keil Software Inc.** 2002. *RTX51 Real-Time Executive for the 8051 Microcontroller*. s.l. : Silicon Valley, 2002.
- Kernighan, Brian W. a Ritchie, Dennis M.** 1998. *The C Programming Language*. Enlwood Cliffs, New Jersey 07632 : PRENTICE HALL, 1998.
- Kopecký, Ladislav.** 2008. *Reluktančný motor a elektromobil*. [PDF dokument] Brno : s.n., 2008.
- NORSONIC, Slovensko s.r.o.** 2008. <http://norsonic.sk>. NORSONIC. [Online] 1. január 2008. [Dátum: 20. október 2011.] www.norsonic.sk/img/Dynamicke_vlastnosti_automobilu.pdf.
- Novák, Vilém.** 2000. *Základy fuzzy modelování*. 1. vyd. Praha : BEN - technická literatura, 2000. ISBN 80-7300-009-1.
- Poliač, F., Fedák, V. a Zboray, L.** 1987. *Elektrické pohony*. Bratislava : Alfa, 1987.
- Richter a Lutz.** 1985. *Betriebssysteme*. s.l. : Teubner Stuttgart, 1985.
- Ripps a David.** 1988. *A Guide to Real-Time Programming*. s.l. : Englewood Cliffs, N.J, Prentice Hall, 1988.
- RTX Real Time Operating System.* **Jadroň, Eduard, Feňo, Ivan a Špánik, Pavol.** 2001. Žilina : 4th International Scientific Conference, 2001.
- Skýva, L.** 1987. *Teória automatického riadenia I*. Bratislava : ALFA, 1987.
- Vegr, Jaromír.** 2009. *Elektromobily - historie a současnost*. <http://www.pro-energy.cz>. [Online] 29. 9 2009. [Dátum: 24. 10 2011.] <http://www.pro-energy.cz/clanky7/3.pdf>.
- Vittek, Ján.** 2004. *Vybrané metody riadenia elektrických pohonov v prostredí MATLAB-Simulink*. Trenčín : Trenčianská univerzita Alexandra Dubčeka, 2004. ISBN 80-8075-039-4.
- Zboray, L., Ďurkovský, F. a Tomko, J.** 2000. *Regulované pohony*. Košice : Viena Košice, 2000. ISBN 80-88922-13-5.
- Zeman, K., Peroutka, Z. a Janda, M.** 2004. *Automatická regulace pohonů s asynchronními motory*. Plzeň : Západočeská univerzita v Plzni, 2004. ISBN 80-7043-350-7.
- Želmíra, Ferková.** 2004. *Bezsnímačové riadenie spínaného reluktančného motora*. AT&P journal. AT&P journal, 2004.

20 Zoznam obrázkov

Obrázok 1, Naplnenie súvislého bloku pamäti pomocou nepriameho adresovania.....	3
Obrázok 2, Príklad správneho zápisu programu v assembleri	3
Obrázok 3, Pamäťový model a mikroprocesora AT89C51RD2 kompatibilného s 8051	4
Obrázok 4, Bloková schéma mikroprocesora AT89C51ED2	5
Obrázok 5, Prerušovacia jednotka mikroprocesora 80C552 v μ Vision4.....	6
Obrázok 6, Prerušovacia jednotka mikroprocesora AT89C2051 v μ Vision4.....	6
Obrázok 7, Vzorový príklad programu v macroassembleri	7
Obrázok 8, Príklad zápisu programu v assembleri A51.....	9
Obrázok 9, Zápis programu v assembleri a jeho pripojenie k jazyku C	11
Obrázok 10, Pripojenie programu v assembleri do jazyka C	11
Obrázok 11, Príklad zápisu programu v assembleri.....	12
Obrázok 12, Preklad programovej knižnice RTX51 Tiny verzie 1.01.....	14
Obrázok 13, Preklad programovej knižnice RTX51 Tiny verzie 2.02.....	14
Obrázok 14, Výpis chybového hlásenia o nepodporovanej 16-bitovej aplikácii.....	15
Obrázok 15, Preklad programu pomocou dávkového súboru	16
Obrázok 16, Výpis prekladu programu v 64-bitovom systéme	16
Obrázok 17, Odladovaný program sériového rozhrania v μ Vision.....	17
Obrázok 18, Nastavenie μ Vision pre presmerovanie sériového kanála na port COM.	18
Obrázok 19, μ Vision3 a využitie funkcie Logic Analyzer - osciloskop	19
Obrázok 20, Nastavenie funkcie Logic Analyzer - osciloskop.....	20
Obrázok 21, Vzorový príklad projektu μ Vision4	25
Obrázok 22, Voľba úborov pre preklad projektu v menu „Project Workspace“	26
Obrázok 23, Voľba mikroprocesora v okne „Options for Target“.....	26
Obrázok 24, Voľba vlastností projektu v okne „Options for Target“	27
Obrázok 25, Nastavenie formátu výstupu prekladaného súboru do HEX v μ Vision3	27
Obrázok 26, Nastavenie optimalizácie a optimalizačného stupňa v μ Vision3	28
Obrázok 27, Použitie najvyšších optimalizačných stupňov v Ax51 a Cx51.....	28
Obrázok 28, Nastavenie integrovaného Debugger-a.....	29
Obrázok 29, Príklad zápisu programu pre Debugger	30
Obrázok 30, Programovací model vývojového nástroja μ Vision4 a vyšší	33
Obrázok 31, Porovnanie veľkostí programu v závislosti na verzii prekladača C51	34
Obrázok 32, Vývoj aplikácie s Cx51 a RTX51.....	35
Obrázok 33, Vývoj aplikácie v prostredí μ Vision4	35
Obrázok 34, Optimalizácia programového kódu kompilátorom Cx51	36
Obrázok 35, Optimalizačné stupne μ Vision	36
Obrázok 36, Deklarácia pamäťovej štruktúry	38
Obrázok 37, Deklarácia pamäťovej štruktúry s určením veľkosti	38
Obrázok 38, Deklarácia jednoduchej štruktúry v C	41
Obrázok 39, Správne uvoľnenie pamäti pomocou <code>os_free_block()</code>	42
Obrázok 40, Nesprávne uvoľnenie pamäti pomocou <code>os_free_block()</code>	42
Obrázok 41, Modifikované prostredie μ Vision4 pre ladenie aplikácií s AGSI ovládačmi.....	43
Obrázok 42, Simulácia LCD display-a AGSI ovládačmi	44
Obrázok 43, Simulácia I ² C s AGSI ovládačmi	44
Obrázok 44, Simulácia stavu procesora s AGSI ovládačmi.....	45
Obrázok 45, Simulácia generátora priebehov s AGSI ovládačmi.....	45
Obrázok 46, Vývoj aplikácie pomocou μ Vision 4.....	46
Obrázok 47, Optimálne rozdelenie pracovnej plochy vývojového nástroja	49

Obrázok 48, Vývojové prostredie μ Vision 2.14 od firmy KEIL pre architektúry 80C51	50
Obrázok 49, Vývojové prostredie μ Vision3 od firmy KEIL pre architektúry 80C51	51
Obrázok 50, Vývojové prostredie μ Vision3 od firmy KEIL pre architektúry 80C166	52
Obrázok 51, Vývojové prostredie μ Vision3 od firmy KEIL pre architektúry ARM7TDMI... ..	53
Obrázok 52, Vývojové prostredie μ Vision3 pre architektúry ARM	54
Obrázok 53, Vývojové prostredie EasyCode 7.1.0.3 od firmy EasyCode Inc. pre 8051	55
Obrázok 54, Moderné možnosti programovania s EasyCODE 9.1.....	56
Obrázok 55, Pracovné prostredie EasyCODE 9.0.0.....	56
Obrázok 56, Vývojové prostredie ProView32	58
Obrázok 57, Vývojové prostredie ADSIM812	59
Obrázok 58, Vývojové prostredie Embedded Workbench.....	60
Obrázok 59, Vývojové prostredie CodeVisionAVR pre architektúry AVR	61
Obrázok 60, Doporučený textový IDE editor k Turbo51.....	62
Obrázok 61, Vývojové prostredie AS552	64
Obrázok 62, Hardware pre mikroprocesory 8051	65
Obrázok 63, Aplikačný model mikroprocesorového systému s 80Cx51	66
Obrázok 64, Vývojová doska s mikroprocesorom Intel 80Cx51	66
Obrázok 65, Aplikačný model mikroprocesorového systému s 80C517	67
Obrázok 66, Vývojová doska s mikroprocesorom Siemens 80C517.....	67
Obrázok 67, Aplikačný model mikroprocesorového systému s 80C166	68
Obrázok 68, Vývojová doska s mikroprocesorom Siemens 80C166.....	68
Obrázok 69, Vývojová doska s mikroprocesorom Philips LPC2129.....	69
Obrázok 70, Vývojová doska s mikroprocesorom Infineon XC167CI.....	69
Obrázok 71, Všeobecný model RTOS	77
Obrázok 72, Bloková schéma systému reálneho času.....	77
Obrázok 73, Mechanizmus prepínania úloh v RTX51.....	78
Obrázok 74, Task management – prepínanie úloh v RTX166.....	79
Obrázok 75, Stack management – alokácia pamäťového priestoru v RTX166	80
Obrázok 76, Stack management – alokácia pamäťového priestoru v RTX51	80
Obrázok 77, Debugging – Stav úloh a sys. prostriedkov RTX166 Tiny a AR166	81
Obrázok 78, Debugging – Stav sputených úloh a systémových prostriedkov RTX51 Tiny ..	81
Obrázok 79, Typy úloh a alokácia pamäťového priestoru v RTX51 Full.....	82
Obrázok 80, Nastavenie premennej RAMTOP pre RTX51 Tiny	85
Obrázok 81, Nesprávne ošetrenie udalosti typu "interval"	88
Obrázok 82, Správne ošetrenie udalosti typu "timeout"	88
Obrázok 83, Komunikácia CAN s fyzickou vrstvou CAN Bus.....	91
Obrázok 84, Formát CAN správ	92
Obrázok 85, Základný koncept CAN	92
Obrázok 86, Spôsob komunikácie aplikácie RTOS a rozhraním CAN	93
Obrázok 87, CAN Controller v μ Vision4	94
Obrázok 88, CAN Message Center v μ Vision4	94
Obrázok 89, Generovanie CAN správ v prostredí μ Vision4	94
Obrázok 90, Otvorený referenčný model zbernice CAN.....	95
Obrázok 91, Vrstvový model zbernice CAN	95
Obrázok 92, Použitie zbernice CAN v automobilovom priemysle.....	96
Obrázok 93, Použitie zbernice CAN v elektrickej trakcii.....	96
Obrázok 94, Rozloženie vývodov zbernice CAN	97
Obrázok 95, Architektúra 8051 – C508 od firmy INFINEON	98
Obrázok 96, Architektúra 8051 – C504 od firmy INFINEON	98

Obrázok 97, Ultra High-Speed 8051 - DS80C400 od firmy Dallas pre sieťové aplikácie	99
Obrázok 98, Ultra High-Speed 8051 – DS80C390 od firmy Dallas s CAN.....	99
Obrázok 99, High-Speed architektúra 8051 – DS89C420 od firmy Dallas Semiconductor..	100
Obrázok 100, High-Speed architektúra 8051 – AT89C51RD2 od firmy ATMEL.....	100
Obrázok 101, High-Speed architektúra 80251 – TSC8x251G2D od firmy ATMEL	101
Obrázok 102, Pamäťový model architektúry 80251 – TSC8x251G2D od firmy ATMEL ...	101
Obrázok 103, Architektúra Tri Core 32 bit. DSP TC1775B od firmy INFINEON	102
Obrázok 104, Architektúra Tri Core 32 bit. DPS TC10GP od firmy INFINEON.....	102
Obrázok 105, Architektúra 32 bit. mikroprocesora TC1775B od firmy INFINEON	103
Obrázok 106, Architektonické vlastnosti architektúry Tri Core od firmy INFINEON	103
Obrázok 107, Bitový invertor portu	111
Obrázok 108, Bitový invertor s EPLD	112
Obrázok 109, Simulácia programu sledovača fáz	119
Obrázok 110, Schéma zapojenia sledovača fáz s AT89LP4052	120
Obrázok 111, Doska plošného spoja sledovača fáz s AT89LP4052.....	120
Obrázok 112, Priame meranie frekvencie mikroprocesorom 80C552 riadený prerušením ...	124
Obrázok 113, Generovanie zapínacích impulzov - simulácia.....	125
Obrázok 114, Zaradenie hodnoty k príslušnosti prvku danej množiny.....	126
Obrázok 115, Fuzzy regulátor v jazyku C s RTX51	127
Obrázok 116, Použitie a zápis overlay-u v μ Vision.....	130
Obrázok 117, Ošetrovanie viacnásobne volaných funkcií v μ Vision5	131
Obrázok 118, Chybové hlásenia pri súčasnom prístupe viacerých úloh.....	132
Obrázok 119, Výpis linkera L51 po kompilácii programu s použitím overlay direktív	132
Obrázok 120, Príklad zápisu ošetrenia nevolanej funkcie	133
Obrázok 121, Výpis programu s chybovým hlásením o nevolanom segmente	133
Obrázok 122, Výpis programu bez chybového hlásenia o nevolanom segmente	133
Obrázok 123, Tvorba adresy E ² PROM pamäti AT24C01 až AT24C16.....	137
Obrázok 124, Použitie E ² PROM pamäti v programe.....	142
Obrázok 125, Vnútoraná schéma DS1807	143
Obrázok 126, Komunikačný model obvodu DS1631 – vnútorné zapojenie.....	148
Obrázok 127, Symbolické priebehy napätí na I ² C rozhraní pri komunikácii	148
Obrázok 128, Detailné priebehy napätí na I ² C rozhraní pri komunikácii	148
Obrázok 129, Časové závislosti na I ² C rozhraní DS1631.....	149
Obrázok 130, Časové závislosti na I ² C rozhraní DS1621.....	151
Obrázok 131, Schéma zapojenia DS1307+.....	153
Obrázok 132, Vnútoraná schéma zapojenia DS1307.....	154
Obrázok 133, Prevod čísla na BCD/DEC formát.....	154
Obrázok 134, Zápis dát v režime Slave Receiver Mode (Blokový mód)	154
Obrázok 135, Čítanie dát v režime Slave Transmitter Mode (Blokový mód)	155
Obrázok 136, Kombinované čítanie dát Slave Receive and Transmit (po byte)	155
Obrázok 137, Komunikačné rutiny pre DS1307 pomocou I ² C.....	156
Obrázok 138, Screenshot obrazovky pool-managementu RTX51 Full	157
Obrázok 139, Správa pamäti v RTX Full cez pool-management	158
Obrázok 140, Simulácia LCD display-a v prostredí Keil μ Vision4	163
Obrázok 141, Schéma zapojenia USB2LPT portu.....	169
Obrázok 142, Doska plošného spoja USB2LPT portu.....	170
Obrázok 143, Generovanie PWM signálu u AT89C4051 pomocou časovača T0.....	171
Obrázok 144, Prevodná tabuľka 7 segmentového displaya v assembleri A51	172
Obrázok 145, Simulácia PWM kanála so 7-segmentovým displayom a LED display-om....	173

Obrázok 146, Simulácia PWM kanála s LED display-om.....	173
Obrázok 147, Zapojenie univerzálneho modulu s DS89C450 s podporou CAN.....	174
Obrázok 148, Doska plošného spoja univerzálneho modulu s podporou CAN.....	175
Obrázok 149, Rozmiestnenie súčiastok univerzálneho modulu s podporou CAN	176
Obrázok 150, ISP pre procesory DS89C450.....	177
Obrázok 151, Pointer v jazyku C	178
Obrázok 152, Implementácia do rtxconf.a51 o nepodporovaný processor DS87C550	181
Obrázok 153, Modifikácia súboru RTXCONF.A51 pre procesor DS87C550	183
Obrázok 154, Náhrada integrálu sumou lichobežníkov	185
Obrázok 155, Náhrada integrálu sumou obdĺžnikov.....	185
Obrázok 156, Stavový diagram polohového PSD regulátora	186
Obrázok 157, Stavový diagram polohového PSD regulátora s filtráciou derivačnej zložky.	186
Obrázok 158, Simulačný model impulzového meniča.....	191
Obrázok 159, Simulačné výsledky modelu impulzového meniča	191
Obrázok 160, Reálne zapojenie impulzového meniča s reguláciou výstupného napätia.....	192
Obrázok 161, Simulačné výsledky reálneho zapojenia impulzového meniča	192
Obrázok 162, Reálne zapojenie impulzového meniča s PFC – Power Factor Correction	193
Obrázok 163, Simulačné výsledky reálneho zapojenia impulzového meniča s PFC.....	194
Obrázok 164, Detailné simulačné výsledky impulzového meniča s PFC.....	194
Obrázok 165, Mostový 3-fázový usmerňovač s napäťovým striedačom a rez. záťažou	195
Obrázok 166, Priebehy napätí a prúdov 3-fázového usmerňovača pri $\alpha=0^\circ$	196
Obrázok 167, Časové priebehy napätí a prúdov v oblasti normálnych zaťažení	196
Obrázok 168, Principiálne generovanie primárnych zapínacích impulzov pre tyristory	198
Obrázok 169, Generovanie impulzov v mikroprocesore 87C552.....	199
Obrázok 170, Vývojová doska diskretného riadenia usmerňovača s 80C552	199
Obrázok 171, Zoznam úloh diskretného regulátora usmerňovača pre tyristory	200
Obrázok 172, Interface riadenia usmerňovača s podpornými obvodmi variantu č.1	200
Obrázok 173, Simulácia regulátora diskretného riadenia s 80C552 varianty č.2	201
Obrázok 174, Doska diskretného riadenia usmerňovača s 80C552 varianty č.2	201
Obrázok 175, Schéma zapojenia diskretného riadenia usmerňovača 80C552 varianty č.2 ...	202
Obrázok 176, Generovanie zapínacích impulzov pre striedač v mikroprocesore 87C552	205
Obrázok 177, Zjednodušený model tyristorového striedača s rezonančným obvodom.....	206
Obrázok 178, Grafické priebehy napätí v tyristorovom striedači s rezonančným obvodom .	206
Obrázok 179, Grafické priebehy napätí v tyristorovom striedači s rezonančným obvodom .	207
Obrázok 180, Detailné priebehy napätí v tyristorovom striedači s rezonančným obvodom .	207
Obrázok 181, Zapínacie impulzy pre jednotlivé diagonály tyristorov	208
Obrázok 182, Schéma zapojenia skalárneho riadenia 3fázového ASM motora	215
Obrázok 183, Bloková schéma skalárneho ASM pohonu s procesorom Infineon XC866	215
Obrázok 184, Výkonová časť 3-fázového striedača ASM.....	216
Obrázok 185, Princíp zmeny napätia na asynchrónnom pohone pomocou tabuľky.....	216
Obrázok 186, Princíp tvorby výstupného sínusového napätia s tabuľkou v pamäti C504	217
Obrázok 187, Ovládací software pre pohon asynchrónneho motora	218
Obrázok 188, Napätie generované metódou SWPWM.....	219
Obrázok 189, Vytvorenie rotujúceho magnetického poľa statora ASM.....	220
Obrázok 190, Schéma riadenia asynchrónneho motora s uzavretou regulačnou slučkou	220
Obrázok 191, Principiálne znázornenie tvorby viacfázovej sústavy.....	221
Obrázok 192, Základné zapojenie statora asynchrónneho motora.....	221
Obrázok 193, Riadenie asynchrónneho motora s SWPWM (Sinusoidal Weight PWM)	222
Obrázok 194, Riadenie asynchrónneho motora pomocou SVM (Space Vector Modulation).....	222

Obrázok 195, Momentová charakteristika asynchrónneho motora.....	223
Obrázok 196, Namerané priebehy napätia na porte mikroprocesora C504	223
Obrázok 197, Rozdelenie elektrických pohonov	226
Obrázok 198, Základný rozdiel medzi SWPWM a SVM	227
Obrázok 199, Detailné znázornenie magnetického obvodu PMSM s jedným pólom	227
Obrázok 200, Spínací plán výkonového meniča s riadením SVM	228
Obrázok 201, Znázornenie rotujúceho vektora magnetického poľa pohonu s SVM.....	228
Obrázok 202, SVM modulácia pre γ ($0^\circ, 30^\circ, 60^\circ$).....	229
Obrázok 203, Vytvorenie rotujúceho vektora elektrického napätia pomocou SVM	229
Obrázok 204, Sektory SVM znázorňujúce priebeh rotácie vektora	230
Obrázok 205, Napäťový vektor pre SVM	231
Obrázok 206, Priebehy napätí na spínacích prvkoch pri $\gamma=0^\circ, 30^\circ, 60^\circ$	232
Obrázok 207, Priebehy napätí prvkoch pri $\gamma=30^\circ$ a vector length=50%, $f_{PWM}=20kHz$	232
Obrázok 208, Priebehy jednotlivých fázových napätí pri SVM modulácii.	233
Obrázok 209, Detailné zobrazenie CAPCOM6 jednotky mikroprocesora C164 a C508.	234
Obrázok 210, Vyťaženie mikroprocesora (CPU Load) pri SVM	235
Obrázok 211, Kaskádne riadenie viacfázového pohonu s SVM.....	236
Obrázok 212, Principiálne riadenie viacfázového pohonu PMSM s SVM.....	236
Obrázok 213, Ladenie a simulácia programu v prostredí μ Vision3	237
Obrázok 214, Schéma zapojenia pohonu s SVM (Space Vector Modulation)	238
Obrázok 215, Výstup mikroprocesora CCx a COUTx s programovateľným dead-time	238
Obrázok 216, Generovanie napätia s CAPCOM6 v móde 1 pre SVM (Center PWM).	239
Obrázok 217, Vývojový diagram riadenia pohonu s SVM (Space Vector Modulation)	239
Obrázok 218, Transformácia napätí do priestorového vektora SSV (Single Space Vector) .	240
Obrázok 219, Grafické znázornenie aproximácie – náhrady vektora $\underline{u}_S$ pomocou $\underline{u}_1$ a $\underline{u}_2$.	241
Obrázok 220, Generovanie napätia pomocou metódy aproximácie vektora $\underline{u}_S$	241
Obrázok 221, Generovanie napätia s modif. SSVM (Symetric Space Vector Modulation)..	242
Obrázok 222, Grafické znázornenie priebehu vektora s modulačným indexom $U > 0.866$.	243
Obrázok 223, Veľkosti napätí pri použití metód SWPWM, SVM a Overmodulation	243
Obrázok 224, Priebehy napätí pri modul. indexe $U < 0.866$ voči neutrálnemu bodu.....	244
Obrázok 225, Priebeh napätia modulovaného pomocou SVM (Space Vector Modulation) .	244
Obrázok 226, Obsah vyšších harmonických zložiek s SVM pri modul. indexe $U \sim 0.866$.	245
Obrázok 227, Ukážka grafického výstupu programu na meranie jednosmerného napätia	250
Obrázok 228, Ukážka priebehu vstupného napätia pre AD prevodník mikroprocesora	250
Obrázok 229, Sekvencie riadiaceho algoritmu aktívneho filtra	252
Obrázok 230, Pamäťový model externej pamäti programu - xBANKING	254
Obrázok 231, Zapojenie pamäti 256kB RAM a EPROM k 8051, 8x32kB.	256
Obrázok 232, Zapojenie 512kB pamäti programu k 8051 pomocou adresovania XDATA ..	257
Obrázok 233, Zapojenie 256kB pamäti programu k 8051 pomocou štyroch 64kB bánk	258
Obrázok 234, Popis premenných (Environments) pre xBANKING 8051	259
Obrázok 235, Zoznam argumentov LX51 a BL51 pre xBANKING 8051	259
Obrázok 236, Ukážka odlad'ovaného programu s RTX51 Full a xBANKING (6 bánk).....	260
Obrázok 237, Nastavenie prostredia KEIL μ Vision 3 pre prácu s xBANKING	261
Obrázok 238, Správa a inštalácia licencií v prostredí μ Vision3	264
Obrázok 239, Zobrazenie licencií v prostredí μ Vision3	264
Obrázok 240, Informačné okno programu μ Vision3	265

21 Zoznam videosekvencií

Video 1, Sledovač fáz s 80C552 riadený prerušením - simulácia.....	120
--	-----

22 Zoznam tabuliek

Tabuľka 1, Zoznam niektorých vektorov prerušení	8
Tabuľka 2, Prehľad SFR registrov mikroprocesora AT89C51RD2 kompatibilného s 8051	8
Tabuľka 3, Charakteristické vlastnosti RTOS od firmy KEIL	71
Tabuľka 4, Časovanie a vlastnosti RTOS AR166 Kernel V1.00 od firmy KEIL	72
Tabuľka 5, Charakteristické vlastnosti AR166 a RTX166Tiny	72
Tabuľka 6, Charakteristické vlastnosti ARTX pre mikroprocesory ARM	72
Tabuľka 7, Prehľad vývojových nástrojov od firmy KEIL	73
Tabuľka 8, Podpora RTOS od firmy KEIL pre rozhranie CAN	93
Tabuľka 9, Charakteristické vlastnosti zbernice CAN	97
Tabuľka 10, Výstupný formát výsledku prevodu z DS1631	150
Tabuľka 11, Výsledok 12-bitového prevodu z DS1631	150
Tabuľka 12, Konfiguračný register DS1631	150
Tabuľka 13, Nastavenie presnosti prevodu teploty v DS1631	150
Tabuľka 14, Vnútorne registre DS1307	153
Tabuľka 15, Vzájomná závislosť medzi uhlom, T_K , T_{K+1} , T_0 , α , β	234
Tabuľka 16, Zoznam funkcií a použitých premenných pre SVM	244
Tabuľka 17, Zoznam funkcií a premenných pre jednotlivé kvadranty SVM	244

23 Zoznam matematických vzťahov

$y_N = y_{N-1} + A \cdot e_N - B \cdot e_{N-1} + C \cdot e_{N-2}$ (1.)	21
$A = K \cdot \left(1 + \frac{\Delta T}{2T_i} + \frac{T_d}{\Delta T}\right)$ (2.)	21
$B = K \cdot \left(1 - \frac{\Delta T}{2T_i} + \frac{2T_d}{\Delta T}\right)$ (3.)	21
$C = K \cdot \left(\frac{T_d}{\Delta T}\right)$ (4.)	21
$TH1 = 256 - 2SMOD \cdot fosc384 \cdot b - s$ (5.)	37
$f = fXTAL24 \cdot 256 \cdot CTH0 + CTL0$ [Hz; Hz, 1, 1] (6.)	124
$y(t) = K \cdot \left[e(t) + \frac{1}{T_i} \int e(t) dt + T_d \frac{de(t)}{dt} \right]$ (7.)	184
$y_N = K \cdot \left[e_N + \frac{1}{T_i} \sum_{i=1}^N \frac{e_i + e_{i-1}}{2} \Delta T + \frac{T_d \cdot (e_N - e_{N-1})}{\Delta T} \right]$ (8.)	184
$y_N = y_{N-1} + \left[K \cdot \left(1 + \frac{\Delta T}{2T_i} + \frac{T_d}{\Delta T}\right) \cdot e_N - K \cdot \left(1 - \frac{\Delta T}{2T_i} + \frac{2T_d}{\Delta T}\right) \cdot e_{N-1} + K \cdot \left(\frac{T_d}{\Delta T}\right) \cdot e_{N-2} \right]$ (9.)	184
$y_N = y_{N-1} + A \cdot e_N - B \cdot e_{N-1} + C \cdot e_{N-2}$ (10.)	184
$A = K \cdot \left(1 + \frac{\Delta T}{2T_i} + \frac{T_d}{\Delta T}\right)$ (11.)	184
$B = K \cdot \left(1 - \frac{\Delta T}{2T_i} + \frac{2T_d}{\Delta T}\right)$ (12.)	184
$C = K \cdot \left(\frac{T_d}{\Delta T}\right)$ (13.)	184
$y_N = y_{N-1} + \left[K \cdot \left(1 + \frac{T_d}{\Delta T}\right) \cdot e_N - K \cdot \left(1 - \frac{\Delta T}{T_i} + \frac{2T_d}{\Delta T}\right) \cdot e_{N-1} + K \cdot \left(\frac{T_d}{\Delta T}\right) \cdot e_{N-2} \right]$ (14.)	186
$y_N = y_{N-1} + \left[K \cdot \left(1 + \frac{\Delta T}{2T_i} + \frac{T_d}{\Delta T}\right) \cdot e_N - K \cdot \left(1 - \frac{\Delta T}{2T_i} + \frac{2T_d}{\Delta T}\right) \cdot e_{N-1} + K \cdot \left(\frac{T_d}{\Delta T}\right) \cdot e_{N-2} \right]$ (15.)	186
$U_L = L \frac{dI_1}{dt}$ (16.)	190
$dI_1 = \frac{U_1 - U_2}{L} \cdot t_1$ (17.)	190
$U_L = L \frac{dI_2}{dt}$ (18.)	190
$-dI_2 = -\frac{U_2}{L} \cdot t_2$ (19.)	190
$U_2 \cdot t_2 = \frac{U_1 - U_2}{t_1}$ (20.)	190

$U_2 = U_1 \cdot \frac{t_1}{t_1 + t_2} = U_1 \cdot \frac{t_1}{T} = U_1 \cdot \delta$	(21.).....	190
$U_1 \cdot I_1 = U_2 \cdot I_2$	(22.).....	191
$I_2 = I_1 \cdot \frac{U_1}{U_2}$	(23.).....	191
$U_d = \frac{3}{\pi} \int_{\frac{\pi}{3} + \alpha}^{\frac{2\pi}{3} + \alpha} \sqrt{6} \cdot U_f \cdot \sin(\nu) d\nu = \frac{3 \cdot \sqrt{6}}{\pi} \cdot U_f \cdot \cos(\alpha)$	(24.).....	197
$U_d = \frac{3 \cdot \sqrt{6}}{\pi} \cdot U_f \cdot \cos(\alpha)$	(25.).....	197
$\delta \approx \frac{T}{t_z} \approx 6.3$	(1; s,s) (26.)	205
$\beta_{el} = 2 \cdot \pi \cdot f_{rez} \cdot t_q$	(rad · s ⁻¹ ; Hz, s) (27.)	208
$\beta_{el} = 360 \cdot f_{rez} \cdot t_q = 360 \cdot 413 \cdot 170 \cdot 10^{-6} = 25,27^\circ$	(°; Hz, s) (28.)	208
$\varpi_1 = \frac{2 \cdot \pi \cdot f}{p}$	(rad · s ⁻¹ ; Hz,1) (29.).....	212
$n_1 = \frac{60 \cdot f}{p}$	(ot · min ⁻¹ ; Hz,1) (30.).....	212
$\Phi = \frac{U_1}{4,44 \cdot f_1 \cdot N_v \cdot k_v} = k \frac{U_1}{f_1}$	(Wb; V, Hz,1,1) (31.)	212
$\frac{U_1}{f_1} = const$	(V, Hz;1) (32.).....	212
$M_M = \frac{3 \cdot U_1^2}{2 \cdot \varpi_1 \cdot X_K} = K_1 \frac{U_1^2}{f_1^2}$	(N · m; V, rad · s ⁻¹ , Ω) (33.)	213
$m = \sum_{i=1}^n \frac{1}{2.048 \cdot i} \cdot f_{PWM}$	(kB; 1, Hz, Hz) (34.).....	213
$U = A \cdot \sin B = \cos(\arccos A) \cdot \sin B$	(35.)	214
$U = \cos A' \cdot \sin B = \frac{1}{2} [\sin(B - A') + \sin(B + A')]$	(36.).....	214
$T_{PWM} = \frac{1}{f_{PWM}} = \frac{1}{20 \cdot 10^3} = 50 \cdot 10^{-6} = 50 \mu s$	(s; Hz) (37.).....	218
$PERIODE = \frac{f_{OSC}}{prescaler \cdot f_{PWM} \cdot 2} = \frac{40 \cdot 10^6}{4 \cdot 20 \cdot 10^3 \cdot 2} = 250$	(1; Hz, Hz) (38.).....	218
$OFFSET = \frac{t_{dead} \cdot f_{OSC}}{prescaler} = \frac{1 \cdot 10^{-6} \cdot 40 \cdot 10^6}{4} = 10$	(1; s, Hz, 1) (39.)	218
$f_{MIN} = \frac{1}{65536 \cdot T_{PWM}} = \frac{1}{65536 \cdot 50 \cdot 10^{-6}} = 0.305 Hz$	(Hz; s) (40.).....	218

$f_{MAX} = \frac{1}{256 \cdot T_{PWM}} = \frac{1}{256 \cdot 50 \cdot 10^{-6}} = 78.125 \text{Hz}$ (Hz; s) (41.)	218
$angle = angle + delta$ (1; 1, 1) (42.)	218
$delta_{MAX_FREQUENCY} = \frac{f_{MAX}}{f_{PWM}} \cdot 65536 = 256$ (1; Hz, Hz) (43.)	218
$delta_{MIN_FREQUENCY} = \frac{f_{MAX}}{f_{PWM}} \cdot 256 = 1$ (1; Hz, Hz) (44.)	218
$\vec{U}_1 = \frac{1}{T_S} \times (T_{K+1} \times V_{K+1} + T_K \times V_K)$ (45.)	231
$T_K = T_S \times \frac{ \vec{U}_1 }{U_{BAT}} \times \sqrt{3} \times \sin\left(\frac{\pi}{3} - \gamma\right)$ (46.)	231
$T_{K+1} = T_S \times \frac{ \vec{U}_1 }{U_{BAT}} \times \sqrt{3} \times \sin(\gamma)$ (47.)	231
$T_0 = T_S(1 - T_{K+1} - T_K)$ (48.)	231
$\underline{u}_S = V_{AN}(t) \cdot e^{j0} + V_{BN}(t) \cdot e^{j\frac{2}{3}\pi} + V_{CN}(t) \cdot e^{-j\frac{2}{3}\pi}$ (49.)	233
$\underline{u}_S = t_a \cdot \underline{u}_1 + t_b \cdot \underline{u}_2$ (50.)	240
$t_a = \frac{2 \cdot U}{\sqrt{3}} \cdot \sin \alpha$ (51.)	240
$t_b = U \cdot \left[\cos(\alpha) - \frac{\sin(\alpha)}{\sqrt{3}} \right]$ (52.)	240
kde $U = \underline{u}_S $ je tzv. Modulačný index, $\alpha = \angle \underline{u}_S$ je uhol vektora napätia (53.)	240
$t_0 = T_0 \cdot (1 - t_a - t_b)$ (54.)	241
$\alpha_1 = \frac{\pi}{6} - \delta$ (55.)	243
$\alpha_1 = \frac{\pi}{6} + \delta$ (56.)	243
$\delta = \arccos\left(\frac{\sqrt{3}}{2 \cdot U}\right)$ (57.)	243
$i_{ZAT} = \frac{a_0}{2} + \sum_{n=1}^{\infty} (a_n \cdot \cos(\omega_n \cdot t)) + \sum_{n=1}^{\infty} (b_n \cdot \sin(\omega_n \cdot t))$ (58.)	251
$I_{1ZAT} = \sqrt{a_1^2 + b_1^2}; \varphi_1 = \arctan\left(\frac{b_1}{a_1}\right)$ (59.)	251
$I_{1SIET} = I_{1ZAT} \cdot \cos \varphi_1$ (60.)	251
$i_{ZAT}(t) = i_{\alpha} + j \cdot i_{\beta}$ (61.)	251

$$C_v = \frac{3}{\pi} \int_0^{\pi/3} RE\{f(\omega t)\} \cdot e^{-j\nu\omega t} \cdot d\omega t + j \frac{3}{\pi} \int_0^{\pi/3} IM\{f(\omega t)\} \cdot e^{-j\nu\omega t} \cdot d\omega t \quad (62.) \dots\dots\dots 252$$

$$C_1 = \frac{3}{\pi} \int_0^{\pi/3} [(RE \cdot \cos(\omega t) + IM \cdot \sin(\omega t)) + j \cdot (IM \cdot \cos(\omega t) - RE \cdot \sin(\omega t))] d\omega t \quad (63.) \dots\dots\dots 252$$

$$C_1 = \frac{1}{N} \left\{ \sum_{i=0}^{N-1} RE(i) \cdot \cos\left(\frac{i \cdot \pi}{3 \cdot N}\right) + \sum_{i=0}^{N-1} IM(i) \cdot \sin\left(\frac{i \cdot \pi}{3 \cdot N}\right) + \right. \\ \left. + \frac{RE(N) \cdot \cos(\pi/3) - RE(0) \cdot \cos(0)}{2} + \right. \\ \left. + \frac{IM(N) \cdot \sin(\pi/3) - IM(0) \cdot \sin(0)}{2} \right\} + \quad (64.) \dots\dots\dots 252$$

$$+ j \frac{1}{N} \left\{ \sum_{i=0}^{N-1} IM(i) \cdot \cos\left(\frac{i \cdot \pi}{3 \cdot N}\right) - \sum_{i=0}^{N-1} RE(i) \cdot \sin\left(\frac{i \cdot \pi}{3 \cdot N}\right) + \right. \\ \left. + \frac{IM(N) \cdot \cos(\pi/3) - IM(0) \cdot \cos(0)}{2} - \right. \\ \left. - \frac{RE(N) \cdot \sin(\pi/3) + RE(0) \cdot \sin(0)}{2} \right\}$$

$$C_1 = a_1 + j \cdot b_1 \quad (65.) \dots\dots\dots 252$$

